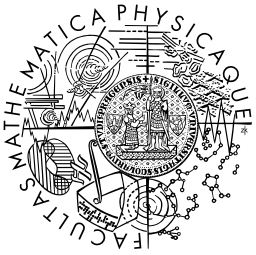


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Tomáš Novotný

Predikce výrobních časů v průmyslu pomocí metod strojového učení

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Umělá inteligence

Praha 2023

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

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

V ..... dne ....

Podpis autora

Především bych chtěl tímto poděkovat mému vedoucímu Martinu Pilátovi za pomoc při tvoření této práce, zejména pak za diskuze vhodnosti jednotlivých pasáží a psychickou podporu k jejímu dokončení. Rovněž bych rád poděkoval mým bývalým kolegům z CIIRC ČVUT, kteří mě s tématem práce seznámili a poskytli mi zdrojová data. V neposlední řadě chci poděkovat mým pracovním kolegům a rodině za ochotu pomoci kdykoli jsem měl speciální požadavky související s tvorbou této práce.

Název práce: Predikce výrobních časů v průmyslu pomocí metod strojového učení

Autor: Tomáš Novotný

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: S rozvojem automatizace plánování v průmyslu se zvyšují požadavky na správný odhad parametrů jednotlivých výrobních operací, především pak jejich časová náročnost. Práce se zabývá různými metodami, jak odhad časové náročnosti nových operací provádět automaticky z dříve provedených operací. První část práce se zaměřuje na standardní regresní algoritmy a jejich úpravu dle vhodnosti pro tento problém. Druhá, hlavní část práce, se soustředí na řešení s využitím hlubokých neuronových sítí, které jsou schopny zpracovat i nestrukturovaná data jako například textový popis operací. Závěrečné výsledky ukazují, že zejména pro zcela nové typy operací dosahuje hluboké učení dobré úrovně predikce. Především ve výrazně dynamických prostředích lze tedy doporučit jeho praktické využití pro plánování nových výrobků.

Klíčová slova: výrobní čas, strojové učení, predikce, hluboké učení

Title: Predicting Production Times Using Machine Learning

Author: Tomáš Novotný

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: With the development of automated task planning in industry, the requirements for a correct estimation of the parameters of individual operations, especially their lead time, are increasing. This thesis is discussing various methods of estimating the lead time for new tasks automatically from previously executed tasks. The first part of the thesis focuses on standard regression algorithms and their modifications according to the suitability for this problem. The second, main part of the theses, focuses on solutions using deep neural networks, which are able to process unstructured data, such as textual descriptions of operations. The final results show that deep learning achieves a good level of prediction, especially for new types of operations. Its practical use can therefore be recommended as an estimate for planning new products, especially in highly dynamic environments.

Keywords: lead time, machine learning, prediction, deep learning

Obsah

Úvod

S rozvojem velké průmyslové výroby se zvyšují nároky na efektivní rozvrhování jednotlivých operací tak, aby se optimalizovalo využití dostupných zdrojů (ať už lidských či strojových) a tím se minimalizovaly výrobní náklady. Pro vytváření rozvrhů je však třeba znát předpokládaný čas, který by měly jednotlivé operace trvat. Tyto časy jsou obvykle pro jednotlivé operace standardizované technologem výroby. Takový přístup je však v řadě hledisek problematický:

- Stanovený výrobní čas ne vždy plně odpovídá realitě,
- výrobní čas pro nové operace není určený předem,
- pro mnoho operací nejsou výrobní časy z kapacitních důvodů určeny vůbec.

Zmíněné problémy lze do určité míry řešit zaměstnáním většího počtu technologů výroby. Levnějším a efektivnějším řešením může však být automatické generování výrobních časů založené na dříve provedených akcích.

V práci se budeme zabývat řešením problematiky automatického generování výrobních časů srovnáním několika různých metod strojového učení, přičemž se pokusíme určovat i spolehlivost predikovaných výrobních časů, je-li to danou metodou možné. Spolehlivost predikce je užitečná informace pro pokročilejší rozvrhovací algoritmy, neboť je pak možné určovat velikost bezpečnostní mezery mezi jednotlivými akcemi v závislosti na jejich důležitosti.

Použité metody budou vyhodnocovány na reálných průmyslových datech. Vzhledem k tomu, že jsou data zadávána lidmi, vzniká v datech velký rozptyl a malé množství výrazně neobvyklých hodnot, tzv. outlierů. Zaměříme se tedy i na jejich detekci a odstranění, neboť pro některé metody mohou taková data výrazně zhoršovat úspěšnost. Cílem práce je však navrhnout obecné postupy použitelné pro jakoukoli standardní průmyslovou výrobu.

Struktura práce

V kapitole 1 si nejprve zavedeme standardizované pojmy používané v problematice výrobních časů, v kapitole 2 se poté zaměříme na další práce zabývající se touto problematikou, jejich výsledky a využitelnost myšlenek pro naše řešení. Kapitola 3 se zabývá strukturou vstupních dat, především pak důsledky plynoucími z jejich reálného původu a výběrem metrik pro hodnocení jednotlivých řešení.

Navrhované modely jsou rozděleny do několika skupin dle typu. V kapitole 4 se nejprve zaměříme na modely vycházející ze standardní lineární regrese a možnosti jejich vylepšení. V kapitole 5 se pak pokusíme nalézt podobnosti v datech a využít je ke zlepšení predikce výrobních časů. V kapitole 6 se zaměříme na modely založené na hlubokých neuronových sítích, které se podobnosti dokážou z dat naučit samy, často s vyšší úspěšností. Nakonec provedeme srovnání jednotlivých modelů a diskuzi výsledků v kapitole 7.

1. Základní pojmy

Nejprve si zaveďme klíčovou terminologii pro zbylou část práce.

1.1 Pojmy plynoucí z průběhu výroby

Průmyslová výroba používá několik standardních pojmů, které využijeme k popisu našich vstupních dat a následné definici řešeného problému.

Definice 1 (Produkt)**.** *Jako* produkt *budeme označovat druh výrobku. Každý produkt má svůj popis (atributy) obsahující například číselné hodnoty či textové řetězce. Množinu všech vyráběných produktů budeme označovat* $\mathcal{P}$*.*

Definice 2 (Kus)**.** *Jako* kus *daného produktu* $p \in \mathcal{P}$ *budeme označovat základní jednotku při výrobě. Obvykle odpovídá jednomu fyzickému výrobku, není to však nezbytné. Může jím být například i relativní hmotnost vyrobené hmoty vzhledem k referenčnímu objektu – konkrétní interpretace je závislá na* p*, přičemž pro naše účely není důležitá.*

Definice 3 (Výrobní operace)**.** Výrobní operací *(zkráceně* operací*) budeme označovat událost, při které je proveden určitý druh výrobního úkonu na několika kusech jednoho produktu* $p \in \mathcal{P}$*. Množinu všech výrobních operací budeme označovat* $\mathcal{V}$*. Ke každé operaci máme rovněž k dispozici dodatečný popis, rozšiřující informace dostupné z popisu samotného produktu.*

Na obsah dostupných atributů pro jednotlivé produkty a výrobní operace a jejich možné využití se zaměříme v kapitole 3.

Definice 4 (Akce)**.** *Zahlášenou* akcí *budeme nazývat trojici* (V, n, t)*, kde* $V \in \mathcal{V}$ *je výrobní operace,* $n > 0$ *je počet kusů, na kterých byla tato operace provedena a* $t > 0$ *je zahlášený čas, neboli oznámený čas trvání této akce. Množinu všech akcí* [1] *budeme označovat* $\mathcal{A}$*.*

Pro zjednodušení značení budeme dále pro $a \in \mathcal{A}$ *označovat její výrobní operaci jako* $V(a)$*, počet kusů jako* $n(a)$ *a zahlášený čas jako* $t(a)$*.*

Množinu $\mathcal{A}$ *můžeme rozdělit na třídy ekvivalence podle jejich operace. Pro danou operaci* V *označíme*

$$\mathcal{A}_V = \left\{ a \in \mathcal{A} \mid V(a) = V \right\}.$$

Povšimněme si, že tato definice umožňuje mít pro jednu výrobní operaci akce s různými počty vyrobených kusů, navíc se v $\mathcal{A}$ může vyskytovat více trojic lišících se pouze časem, zahlášené časy tedy zjevně nejsou deterministické. Rovněž může pro jeden produkt $p \in \mathcal{P}$ existovat více operací $V \in \mathcal{V}$, které byly na p prováděny, na jejich rozlišení tedy nebude možné využít informace o produktu p ale pouze informace z tabulky 3.2.

[1] Ve vstupních datech se jedna akce (V, n, t) může vyskytovat vícekrát, formálně by se tedy měla takto definovaná $\mathcal{A}$ označovat jako multimnožina. Můžeme však místo trojic uvažovat čtveřice (ID, V, n, t), kde přidané ID je unikátní pořadové číslo. Tyto čtveřice již tvoří množinu a jelikož pořadové číslo nenese žádnou informaci (odpovídá pouze pořadí v datech a není nijak spojeno s časem provedení akce), budeme ho implicitně vynechávat.

1.2 Problematika výrobních časů

Pro každou operaci chceme zavést standardizované časy, které by měly jejich akce v průměru trvat. Standardně se pro každou operaci V uvažuje lineární model jejího trvání, tedy že by bez šumu měly zahlášené časy akcí v A_V být lineárně závislé na počtu jimi vyrobených kusů, viz například Polgar (1996).

Definice 5 (Výrobní čas)**.** *Definujme funkci* $T : \mathcal{V} \to \left(\mathbb{R}_0^+\right)^3$*. Hodnotu* $T(V) = (t_b, t_a, t_e)$ *budeme označovat jako* výrobní čas *operace* V*, kde* t_b*,* t_a *a* t_e *značí po řadě přípravný čas, jednicový čas a ukončovací čas operace* V*.*

Obdobně jako u akcí budeme pro zjednodušení označovat přípravný čas, jednicový čas a ukončovací čas $T(V)$ *po řadě jako* $t_b(V)$*,* $t_a(V)$ *a* $t_e(V)$*.*

Výrobní čas v průmyslu tedy odpovídá času, který by měl být potřeba pro provedení akce s danou operací – přípravný čas odpovídá přípravě před provedením operace, jednicový čas odpovídá času na výrobu jednoho kusu a ukončovací čas odpovídá času nutnému k uzavření operace.

Tento model výrobního času vychází z reálné výroby, lze však předpokládat, že nebude zcela přesný. Jednicový čas pro některé operace může se zvyšujícím se počtem zároveň vyráběných kusů kolísat, například je-li k výrobě použita forma schopna pojmout pevný počet kusů. Vzhledem k běžné kvalitě dostupných dat nicméně jeho jednoduchost výrazně omezuje možné přeučování použitých metod.

Definice 6 (Predikovaný čas)**.** *Nechť* $a \in \mathcal{A}$ *je zahlášená akce a nechť* $T\left(V\left(a\right)\right)$ *je výrobní čas odpovídající této akci.* Predikovaným časem *pro tuto akci a poté budeme označovat hodnotu* $t_{pred}(a) = t_b\left(V(a)\right) + t_a\left(V(a)\right) n(a) + t_e\left(V(a)\right)$*, kde* $t_{pred} : \mathcal{A} \to \mathbb{R}^+$ *přiřazuje každé akci* $a \in \mathcal{A}$ *odhad hodnoty* $t(a)$ *založený na zvoleném modelu výrobních časů* T*.*

Po predikovaných časech bychom chtěli, aby co nejlépe odpovídaly akcím, tedy aby zvolený model byl co nejblíže skutečným výrobním časům. Na postup jejich hledání se zaměříme v kapitole 3 zabývající se analýzou vstupních dat, konkrétně v sekci 3.2.

Jelikož u t_b a t_e záleží pouze na jejich součtu, budeme při predikcích uvažovat vždy $t_e = 0$ a namísto t_b budeme uvažovat součet reálného přípravného a ukončovacího času.

2. Související práce

Řešený problém můžeme rozdělit na tři základní části:

- Určování výrobních časů,
- Regresní algoritmus schopný pracovat s komplexní strukturou nezávislých proměnných,
- Čištění vstupních dat a výběr pouze atributů užitečných pro učení.

V této kapitole se zaměříme zejména na práce zabývající se určováním výrobních časů s případným přesahem do zbylých částí. Existence práce, která by se obecně zabývala automatickou predikcí na základě reálných dat ovšem není známa.

Obě zbylé části jsou do určité míry závislé na struktuře dat a použitém algoritmu, budeme se jimi tedy zabývat až v dalších kapitolách dle jejich relevance k jednotlivým metodám.

2.1 Metody určování výrobních časů

Nejčastější způsob určování výrobních časů je využití technologa výroby zmíněného v úvodu. Naším cílem je však dosažení (alespoň částečné) automatizace – tradiční variantou jsou knihy s časovými tabulkami určujícími standardizované hodnoty dle použitého materiálu, typu výrobku a jeho rozměrů, prováděné operaci apod.

2.1.1 Manuálně vytvořené tabulky pro predikci

Dobrým zdrojem informací pro mechanické stroje jako jsou například svěráky, obráběcí stroje či pily je práce „Simplified time estimation for basic machining operations“ (Polgar, 1996), jež byla následně vydána i jako samostatná brožura. Tato práce obsahuje tabulky časů pro řadu typů strojů, včetně konkrétních příkladů s jejich dobovou cenou. Výrobní časy jsou zde děleny podrobněji:

- Čas nastavení stroje – čas potřebný k přípravě stroje před použitím materiálu; přibližně odpovídá t_b,
- Čas upevnění dílu – interval od vzetí materiálu po jeho plnou přípravu ke spuštění stroje; většina je součástí t_a, část může být i t_b (např. některé operace mohou být jednorázové),
- Rychlost zpracování – např. rychlost obrábění v závislosti na materiálu; obvykle tvoří většinu t_a,
- Čas odebrání materiálu a výrobku – částečně spadá pod t_a i t_e, dle toho, zda se určitý krok opakuje pro každý díl, či jen na konci výroby.

U některých strojů s vyměnitelnými či nastavitelnými částmi jsou zde rovněž uvedeny časy pro přenastavení mezi jednotlivými operacemi, které se mohou významně lišit od plného nastavení stroje. Celkový predikovaný čas dle této práce je pak součtem hodnot z odpovídajících tabulek podle typu operací.

Tato a podobné práce jsou častým základem pro technology výroby. Obsažené tabulky jsou velmi podrobné a jsou do určité míry schopny pracovat i se strukturovanými atributy operací jako například použitý materiál či rozměry produktu. Pro definované situace tedy mohou sloužit jako dobrá predikce. Nevýhodou je však omezené využití pro chybějící druhy materiálů, výrobních parametrů či typů operací. V takových případech lze využít hodnoty pro nějakou konfiguraci podobnou jiné, v práci definované, ovšem přesnost takové predikce s jejich rozdílností rychle klesá.

2.1.2 Predikce za pomoci dolování z dat

Namísto manuálně vytvořených tabulek bychom raději chtěli využít možnost automatického určení výrobních časů, což je i hlavním tématem této práce. Pokročilý přístup k tomuto problému lze nalézt v článku „Manufacturing lead time estimation using data mining“ (Öztürk a kol., 2005).

Článek se zabývá predikcí celkových výrobních časů (lead time) jednotlivých výrobků v několika simulovaných typech výrobních linek se strukturou založenou na analýze reálných provozů dle autorů Ruben a Mahmoodi (2000). Na rozdíl od naší práce ovšem pro každý výrobek předpokládají známý jeho skutečný přípravný i jednicový čas všech operací, které na něm budou provedeny. Článek navíc uvažuje existenci pouze 6 různých typů operací, 10 typů výrobků a každý typ má náhodně, ovšem pevně vybranou sekvenci operací. Za celkový výrobní čas daného výrobku se poté považuje součet časů všech na něm prováděných operací a vyčkávání mezi operacemi v případě obsazeného stroje – neznámý rozptyl výrobních časů je tedy způsoben čistě čekacími dobami.

Diskutovány jsou 3 hlavní typy výrobních linek:

- Typ V: první operace všech typů výrobků je prováděna na shodném zdroji,
- Typ A: poslední operace všech typů výrobků je prováděna na shodném zdroji,
- Typ I: operace jednotlivých typů výrobků a jejich pořadí jsou zvoleny rovnoměrně náhodně.

Typ V je obecně o něco jednodušší na predikci, neboť první operace je často bottleneck a přesně známe čas, kdy výrobek k této operaci dorazí. Typ A se může zdát obdobný. Jeho nevýhodou však je, že predikce pro jeho bottleneck je závislá na přesnosti odhadu časů doražení jednotlivých výrobků k této operaci. Typ I neurčuje žádné podmínky, proto je obecně nejnáročnější provést pro něj dobrou predikci vzhledem k dynamickému střídání jednotlivých výrobků.

Cílem článku je nalézt dobrý odhad celkového výrobního času pro všechny přicházející požadavky na výrobky (přicházejících v náhodných intervalech s exponenciálním rozdělením) na základě jejich vlastností jako jsou počet kusů, výrobní čas každé z jeho operací, statistika výrobních časů několika předchozích

požadavků a informace o aktuální zaplněnosti strojů nezbytných ke zpracování nového požadavku.

K řešení problému jsou využité regresní stromy založené primárně na práci „Classification and Regression Trees“ (Breiman, 1984), přičemž zde je třeba provádět diskretizaci jednotlivých atributů kvůli jejich spojitosti. K experimentům využili autoři program Cubist (Quinlan, 2003), který je schopný na základě zadaných parametrů provést výběr vhodných atributů i vybrat optimální hodnotu pro porovnání do každého vrcholu vytvářeného regresního stromu. Diskuze výběru vhodných parametrů je založena na informační hodnotě jednotlivých atributů a hledání hranice mezi podtrénováním a přetrénováním modelu.

Výsledky ukazují schopnost modelu dobře predikovat celkový výrobní čas. Naměřená relativní chyba predikce pro výrobní linku typu V je rovna 0,18, pro typ A rovna 0,27 a pro typ I (se stejnými parametry) rovna 0,38. Výrazně lepší predikce pro typ V je v souladu s očekáváním výše.

Hlavní nevýhodou tohoto článku je zaměření na jediný typ algoritmu a použití uměle vygenerovaných dat, zejména pak malé počty typů výrobků a operací a předpoklady pevných distribucí hodnot jednotlivých atributů výrobků. Problémy reálných dat tak řeší pouze částečně – nebylo třeba nijak řešit extrémní outliery v nezávislých i závislých proměnných způsobené vnějším faktorem a výrobní časy jednotlivých kroků jsou rovněž známé. Atributy jsou omezeny na číselné typy, nijak tedy neřeší komplexnější informace jako například znakové řetězce. Jeho výhodou je diskuze problému zaměřená na dolování z dat, zejména pak vliv různých situací na distribuce jednotlivých atributů.

3. Analýza a zpracování vstupních dat

Jak již bylo zmíněno v úvodu, zdrojová data použitá pro učení pocházejí z reálné průmyslové výroby. Data jsou tvořena sadami:

- Operací $\mathcal{V}$,
- Akcí $\mathcal{A}$ a
- Produktů $\mathcal{P}$.

Pro každou operaci a produkt máme k dispozici jejich strukturovaný popis. V této kapitole zanalyzujeme jejich obsah a provedeme diskuzi řešení možných problému predikce.

3.1 Struktura zdrojových dat

Pro účely práce máme k dispozici celkem 22364 produktů, se kterými je prováděno celkem 42769 různých operací, pro které bylo provedeno celkem 128458 akcí. Pro část produktů nebyly přímo prováděné žádné operace, takové produkty slouží jako definice materiálů využitých pro odpovídající produkty.

Každý produkt obsahuje sadu informací popsaných v tabulce 3.1, přičemž rozměry (výška, šířka, hloubka) a typ materiálu se vyskytují pouze u materiálů. Každá operace obsahuje informace dle tabulky 3.2. Položka „zdrojové materiály“ v tabulce 3.1 vždy obsahuje také všechny zdrojové materiály všech specifikovaných materiálů, neexistují tedy $p_1, p_2, p_3 \in \mathcal{P}$ takové, že p_3 je zdrojovým materiálem p_2, p_2 je zdrojovým materiálem p_1, ale p_3 není zdrojovým materiálem p_1.

Použitá data byla anonymizována, lze je nalézt v příloze A.1. Jednotlivé kategorie, případně znaky pro textové řetězce, byly převedeny na unikátní číselné identifikátory. Tato úprava nemá vliv na použité algoritmy, neboť pro kategorie využíváme pouze porovnávání na rovnost, pro řetězce pak typy znaků a jejich posloupnost v jednotlivých řetězcích. Všechny tyto vlastnosti byly po anonymizaci zachovány – speciální identifikátory odpovídající prázdným kategoriím či bílým znakům zůstaly známy.

Informace	Typ	Výskyt	Pokrytí akcí	Pokrytí materiálů akcí
Název produktu	řetězec	90,8%	95,2%	99,2%(98,8%)
Alternativní název	řetězec	3,3%	6,1%	0,9%(0,9%)
Typ produktu[1]	množ. kategorií	32,6%	71,2%	8,5%(24,6%)
Výška	kladné číslo	1,2%	0%	89,6%(38,6%)
Šířka	kladné číslo	1,6%	0%	90,6%(39,6%)
Hloubka	kladné číslo	1,6%	0%	89,4%(39,0%)
Typ materiálu[2]	kategorie	0,8%	0%	21,3%(9,1%)
Zdrojové materiály	množ. produktů	87,1%	99,2%	9,6%(27,9%)[3]

Tabulka 3.1: Zdrojová data k jednotlivým typům produktů.

Informace	Typ	Výskyt	Pokrytí akcí
Výrobní pracoviště	kategorie	100%	100%
Typ operace	kategorie	100%	100%
Poznámka	řetězec	21,4%	30,9%
Typ výrobku	produkt	100%	100%

Tabulka 3.2: Zdrojová data k jednotlivým operacím.

3.1.1 Zastoupení jednotlivých typů informací v datech

Jednotlivé položky nemusejí být v datech vždy uvedeny – třetí sloupec v tabulkách 3.1, resp. 3.2 odpovídá relativnímu zastoupení vyplněných hodnot vzhledem k $\mathcal{P}$, resp. $\mathcal{V}$, ve čtvrtém pak jejich výskyt vzhledem k $\mathcal{A}$.

Malá část produktů slouží pouze jako materiály – tyto materiály nejsou přímo vyráběny (nejsou tedy produktem žádné operace), obsahují však často dodatečné informace jako například rozměry či typ materiálu. Z tohoto důvodu uvádíme v tabulce 3.1 dvě dodatečné hodnoty. První udává, jaká část akcí z $\mathcal{A}$ patří k produktu, jehož *alespoň jeden* zdrojový materiál obsahuje danou informaci. Hodnota v závorce pak odpovídá výskytu mezi všemi materiály všech akcí (tj. všechny uspořádané dvojice (a, p), kde $a \in \mathcal{A}$ a $p \in \mathcal{P}$ takový, že je zdrojovým materiálem produktu odpovídajícímu a), jichž je celkem 331871.

Můžeme si povšimnout zvláštnosti, že přestože rozměry jsou přítomny ve zdrojových materiálech přibližně v 90% produktů akcí a typ produktu je přítomen pouze v 8,5% (rozdíl je tedy 81%), při uvážení všech materiálů přes všechny akce je to již 39% a 24,6% (rozdíl je tedy jen 14%). Toto chování je způsobeno malou částí produktů, které jsou složeny z mnoha podproduktů označených jako jejich zdrojové materiály. Většina z nich jsou však běžně vyráběné produkty na kterých jsou prováděny operace, mají tedy definován typ produktu, ale nemají rozměry.

3.1.2 Rozložení zahlášených akcí

Rozsah časů trvání a počtu vyrobených kusů je ve vstupních datech vysoký, jak můžeme vidět na obrázku 3.1 – pro většinu dat se v obou osách pohybují hodnoty v řádech od 10^0 po 10^3. Zároveň je mezi časem a počtem kusů pouze nízká korelace ($R^2 = 0{,}013$), bez dodatečných informací o jednotlivých operacích tedy není predikce možná.

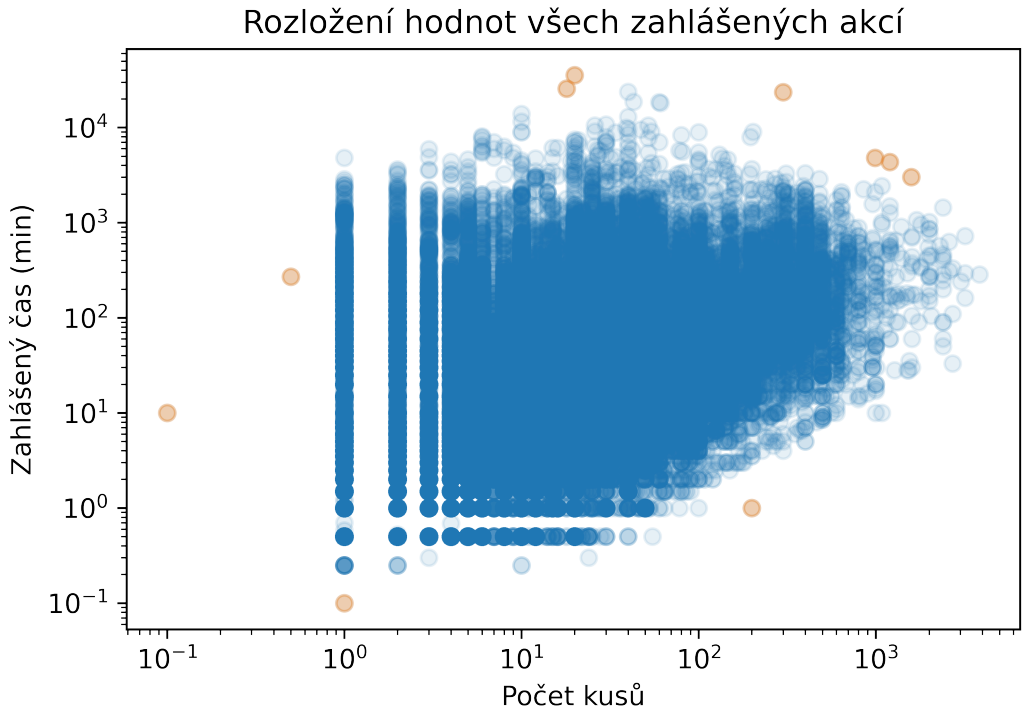
Pokud se však podíváme na jednotlivé typy operací, zjistíme, že rozložení hodnot jejich akcí je obvykle mnohem méně náhodné. Příklad tří vybraných typů 20, 29 a 38 lze nalézt na obrázku 3.2. Provedením lineární regrese zjistíme, že jejich průsečíky s časovou osou (tedy očekávaná t_b) jsou po řadě 370, 36 a 11, směrnicemi (tedy očekávanými t_a) 2,52, 0,56 a 0,06 a R^2 rovným 0,0013[4], 0,29

[1]Materiály mají často typ produktu znamenající „nedefinovaný“ místo prázdné hodnoty.

[2]Tato hodnota je přítomna u všech produktů, ovšem typ s číslem 25 odpovídá nedefinovanému typu. Jelikož má tuto hodnotu 99,2% produktů, budeme ho v této statistice považovat za prázdný.

[3]71% akcí je provedeno na produktech s právě jedním materiálem.

[4]Nízký korelační koeficient operace typu 20 je způsoben zejména několika akcemi s extrémními hodnotami. Na tento problém se zaměříme v sekci 3.3



Obrázek 3.1: Rozložení zahlášených hodnot všech akcí z $\mathcal{A}$ s vyznačenými příklady outlierů

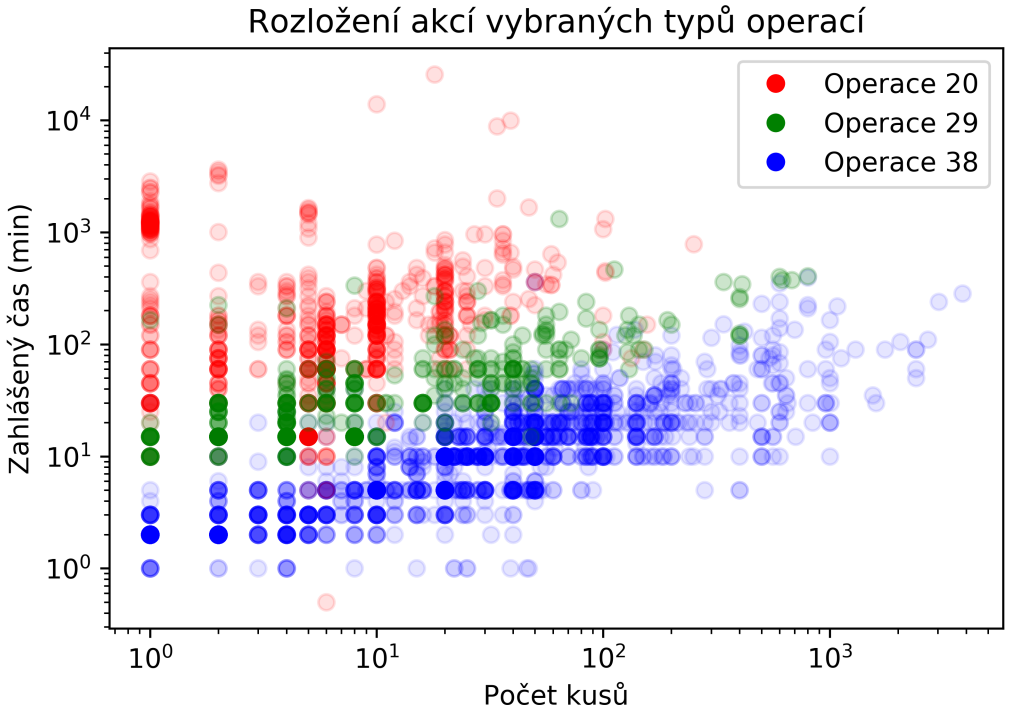
a 0,29. Toto zjištění je ve shodě s očekáváním, zejména pro typy 29 a 38 – například různé výrobky vyřezávané laserem by měly vykazovat podobné hodnoty výrobních časů, zejména pak nízký jednotkový čas kvůli strojům schopným současně vyřezat desítky kusů. Pokud tedy nejsme z dat schopni provést spolehlivou predikci pro konkrétní operaci, je použití informací o jejím typu často přijatelnou náhradou, kterou budeme využívat zejména pro algoritmy v kapitole 4.

3.2 Volba hodnotící funkce

Klíčovým prvkem je volba pevné chybové funkce err, která by měla co nejlépe vystihovat realitu. Tato funkce je nezbytná pro porovnávání úspěšnosti jednotlivých modelů použitých pro hledání skutečných výrobních časů T.

Chybovou funkci $err : \left(\mathbb{R}^{+}\right)^{2} \to \mathbb{R}$ vybereme na základě několika pozorování plynoucích z dat a procesu jejich získávání:

1. Při shodě zahlášeného a predikovaného času by měla být hodnota err rovna 0.
2. Funkce err by měla být spojitá, pravděpodobně i diferencovatelná. Nedává smysl, aby pro nějakou kombinaci zahlášeného a predikovaného výrobního času jejich libovolně malá změna způsobila skokovou změnu hodnoty err.
3. Různé náhodné efekty ovlivňující jednotlivé akce pro konkrétní operaci způsobují, že akce můžeme aproximovat jako vzájemně nezávislé. Ideálně bychom tedy chtěli, aby očekávané distribuce hodnot err akcí zahlášených pro jednotlivé operace přibližně odpovídaly normálním rozdělením (se střední hodnotou 0, vzhledem k pozorování 1).



Obrázek 3.2: Rozložení zahlášených hodnot akcí třech vybraných typů operací

4. jelikož efekty měnící čas trvání akcí jsou obvykle stabilní v průběhu celé akce (zkušenosti pracovníka, stav stroje apod.), měla by být hodnota *err* závislá pouze na relativní odchylce času trvání akce od očekávaného výrobního času,

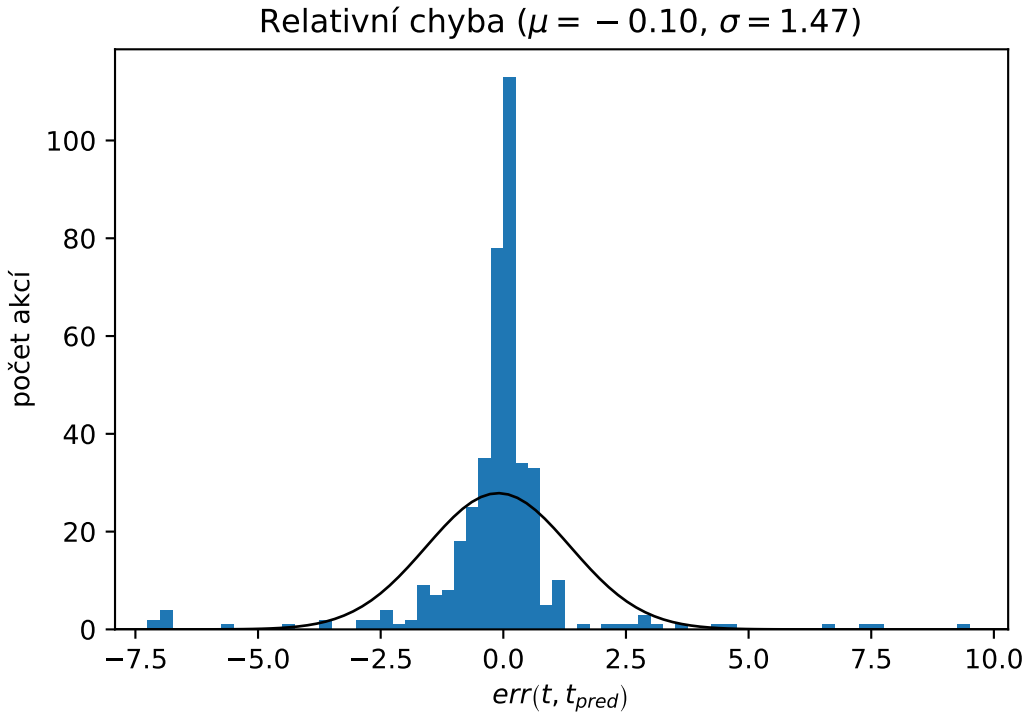
5. Klíčové zdroje chyb jsou nezávislé a navzájem aditivní – například koeficienty z přidání dodatečného kroku do prováděné akce, rychlosti použitého stroje a zkušenosti pracovníka se chovají jako by na zahlášený čas byly aplikovány postupně.

Poznámka. Pozorování 5 nemusí být vždy vždy zcela pravdivé, je však považováno za dobrou aproximaci, viz například článek o distribucích výrobních časů (Dudley, 1963) či kniha „Simulation Modeling and Analysis“ (Law a Kelton, 2000). Rozptyl hodnot v našich datech rovněž ukazuje, že pro časté operace se zahlášené časy chovají výrazně blíže logaritmicko-normálnímu rozdělení (které je důsledkem těchto pozorování, což dokážeme ve větě 3) než například použití pouze poměru zahlášeného a predikovaného výrobního času, viz obrázky 3.3 a 3.4. To je způsobeno zejména častým výskytem zahlášených časů s několikanásobnou hodnotou oproti očekávané, které lze poměrně dobře pomocí pozorování 5 vysvětlit.

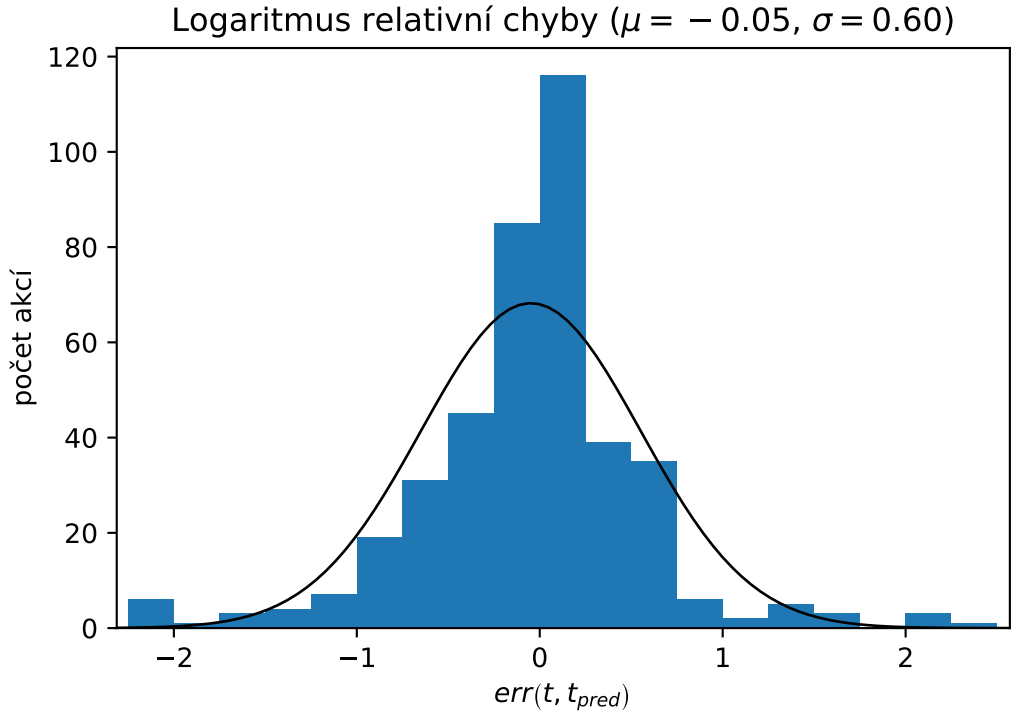
Zapišme si exaktně obsahy jednotlivých pozorování:

1.
$$\forall x \in \mathbb{R}^+ : err(x, x) = 0$$

2.
$$\forall x, y, \delta \in \mathbb{R}^+ \exists \epsilon \in \mathbb{R}^+ \forall z \in (-\varepsilon, \varepsilon) : \Big|err(x + z, y) - err(x, y)\Big| < \delta,$$



Obrázek 3.3: Rozložení hodnot err s odpovídajícím normálním rozdělením $\mathrm{N}\left(\mu, \sigma^2\right)$ pro 5 operací s největším počtem akcí, pokud by pro ni byla použita relativní chyba $err(x, y) = \frac{y-x}{\min(x,y)}$



Obrázek 3.4: Rozložení hodnot err s odpovídajícím normálním rozdělením $\mathrm{N}\left(\mu, \sigma^2\right)$ pro 5 operací s největším počtem akcí, pokud by pro ni byl použit logaritmus relativní chyby $err(x, y) = \ln(y) - \ln(x)$

$$\left|err(x, y+z) - err(x, y)\right| < \delta$$

3. $\forall V \in \mathcal{V}$ by měla distribuce $err\left(t(a), t_{pred}(a)\right)$ všech $a \in \mathcal{A}_V$ přibližně odpovídat $\mathrm{N}\left(0, \sigma^2\right)$ pro nějaké $\sigma > 0$, neboli

$$\forall V \in \mathcal{V}, \forall a \in \mathcal{A}_V : P\left[err\left(t(a), t_{pred}(a)\right) = x\right] \approx \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}$$

4.
$$\forall x, y \in \mathbb{R}^+ : err\left(x, y\right) = err\left(1, \frac{y}{x}\right)$$

5.
$$\forall x, y, z \in \mathbb{R}^+ : err\left(x, y\right) + err\left(y, z\right) = err\left(x, z\right)$$

Povšimněme si, že pozorování 1 je přímým důsledkem pozorování 5 při dosazení x, x, y za x, y, z, není tedy pro důkazy nezbytné. Zbývá tedy určit, jaké funkce err splňují všechny naše požadavky. Klíčovým bodem bude dokázání, že

$$\forall r \in \mathbb{R}^+ : err\left(1, r^x\right) = x \cdot err\left(1, r\right) \tag{3.1}$$

platí pro libovolné $x \in \mathbb{R}$.

Lemma 1. *Rovnost 3.1 platí pro všechna celá čísla, neboli*

$$\forall r \in \mathbb{R}^+, n \in \mathbb{Z} : err\left(1, r^n\right) = n \cdot err\left(1, r\right).$$

Důkaz. Nejprve indukcí ukážeme, že rovnost 3.1 platí pro všechna celá nezáporná čísla. Pro $n = 0$ toto tvrzení plyne přímo z pozorování 1 při dosazení $x = 1$, pro $n = 1$ zřejmě platí.

Předpokládejme nyní, že tvrzení platí pro $n - 1$. Ukážeme, že pak nutně platí i pro n. Nejprve využijme pozorování 5 a dosaďme 1, r^{n-1} a r^n za proměnné x, y a z:

$$err\left(1, r^{n-1}\right) + err\left(r^{n-1}, r^n\right) = err\left(1, r^n\right)$$

Druhý člen součtu můžeme upravit dle pozorování 4, čímž získáme

$$err\left(r^{n-1}, r^n\right) = err\left(1, \frac{r^n}{r^{n-1}}\right) = err\left(1, r\right)$$

Z indukčního předpokladu tedy

$$(n-1)\, err\left(1, r\right) + err\left(1, r\right) = err\left(1, r^n\right)$$

$$n \cdot err\left(1, r\right) = err\left(1, r^n\right)$$

Zbývá ukázat platnost lemmatu i pro záporná celá čísla. Dosadíme-li do pozorování 5 za proměnné x, y, z, hodnoty $1, x, 1$ dostaneme

$$err\left(1, x\right) + err\left(x, 1\right) = err\left(1, 1\right) = 0,$$

$$err\left(x, 1\right) = -err\left(1, x\right)$$

Pro každé kladné x. Z pozorování 4 poté plyne

$$err\left(1, x^{-1}\right) = -err\left(1, x\right)$$

a dosazením r^n místo x

$$err\left(1, r^{-n}\right) = -err\left(1, r^n\right) = -n \cdot err\left(1, r\right)$$

□

Lemma 2. *Rovnost 3.1 platí pro všechna reálná čísla.*

Důkaz. z lemmatu 1 již víme, že rovnost 3.1 platí pro všechna celá čísla. Vezměme libovolné racionální číslo $\frac{a}{b}$ pro $a \in \mathbb{Z}, b \in \mathbb{N}$. Dosadíme-li do rovnosti 3.1 $r^{\frac{1}{b}}$ za r a b za x, dostaneme

$$err\left(1, r^{\frac{ab}{b}}\right) = b \cdot err\left(1, r^{\frac{a}{b}}\right),$$

$$err\left(1, r^{\frac{a}{b}}\right) = \frac{err\left(1, r^a\right)}{b} = \frac{a}{b} \cdot err\left(1, r\right).$$

Rovnost 1 tedy platí i pro všechna racionální čísla. Bohužel, z pozorování 4 a 5 již nelze rovnost 3.1 dokázat i pro iracionální čísla (viz poznámka níže). Dle pozorování 2 však musí být *err* spojitá, což k jednoznačnosti stačí.

Předpokládejme pro spor, že pro nějaké iracionální x rovnost 3.1 neplatí (tedy $\left|err\left(1, r^x\right) - x \cdot err\left(1, r\right)\right| = \delta'$ pro nějaké $\delta' > 0$). Pro každé $\epsilon' < \frac{\delta'}{2err(1,r)}$ [5] v intervalu $(x - \varepsilon', x + \varepsilon')$ pak můžeme nalézt nějaké racionální q, pro které

$$\left|err\left(1, r^q\right) - err\left(1, r^x\right)\right| = \delta' - \left|\left(x - q\right) err\left(1, r\right)\right| > \frac{\delta'}{2}.$$

Pro $\delta < \delta'$ by tedy nebylo možné nalézt ε splňující pozorování 2, což je spor s předpokladem.

□

Poznámka. Pokud si označíme $f(x) := err\left(1, r^x\right)$ pak lze pozorování 5 přepsat při dosazení $1, r^x, r^{x+y}$ za x, y, z a použití pozorování 4 na druhý člen součtu na $f(x) + f(y) = f(x + y)$ platné pro každé reálné x a y. To odpovídá známé Cauchyho funkcionální rovnici, jejíž všechna řešení za určitých předpokladů (například spojitosti f alespoň v jednom bodě) odpovídají $f(x) = f(1) \cdot x$. Bez dodatečných podmínek (viz Hamel, 1905) lze ovšem s pomocí axiomu výběru nalézt i nespojitá řešení.

Nyní již máme dokázáno klíčové tvrzení pro popsání všech možných řešení *err*:

Věta 3. *Jedinými funkcemi splňujícími všechna pozorování jsou*

$$err(x, y) = c \cdot \left(\ln\left(y\right) - \ln\left(x\right)\right)$$

pro nějaký parametr $c \in \mathbb{R}$.

[5] Předpokládejme $err\left(1, r\right) \neq 0$, jinak by *err* byla ve všech racionálních číslech rovna 0 a spor je triviální.

Důkaz. Dosaďme do pozorování 5 za proměnné x, y, z, hodnoty $1, x, y$ pro libovolná $x, y \in \mathbb{R}^+$.

$$err\,(1,x) + err\,(x,y) = err\,(1,y)\,.$$

Jelikož jsou x i y kladné, můžeme ekvivalentně psát

$$err\left(1, e^{\ln(x)}\right) + err\,(x,y) = err\left(1, e^{\ln(y)}\right),$$

$$err\,(x,y) = err\left(1, e^{\ln(y)}\right) - err\left(1, e^{\ln(x)}\right).$$

Dle Lemmatu 2 platí $err\left(1, e^{\ln(x)}\right) = \ln(x) err\,(1,e)$ pro každé $x \in R^+$, dosazením tedy získáme

$$err\,(x,y) = err\,(1,e)\ln(y) - err\,(1,e)\ln(x) = c \cdot \Big(\ln(y) - \ln(x)\Big)$$

pro parametr $c = err\,(1,e)$, který můžeme libovolně nastavit.

Dosazením můžeme snadno ověřit, že všechna nalezená řešení splňují pozorování 1, 2, 4 a 5. Dle grafu 3.4 je se vhodným c rovněž dobrou aproximací pozorování 3 pro reálná data.

□

Zbývá nám odvodit, co by měl splňovat dobrý model skutečných výrobních časů T a jak tedy budeme jednotlivé modely porovnávat. K tomu využijeme metodu hledání nejvěrohodnějšího odhadu.

Dle pozorování 3 můžeme předpokládat, že pro každou akci a bude platit

$$P\left[err\left(t(a), t_{pred}(a)\right) = x\right] = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{x^2}{2\sigma^2}},$$

kde σ je neznámá pevná standardní odchylka rozložení err. Vzhledem k předpokladu nezávislosti jednotlivých akcí je poté nejvěrohodnější odhad roven součinu

$$\underset{t_a(V), t_b(V)}{\arg\max}\, P\left[err\left(t\,(a_1), t_{pred}\,(a_1)\right) = x_1, err\left(t\,(a_2), t_{pred}\,(a_2)\right) = x_2, \dots\right] =$$

$$= \underset{t_a(V), t_b(V)}{\arg\max} \prod_{a \in \mathcal{A}} P\left[err\left(t\,(a), t_{pred}\,(a)\right) = x\right].$$

Dosazením za err a zlogaritmováním celé rovnosti [6] dostaneme

$$= \underset{t_a(V), t_b(V)}{\arg\max} \prod_{a \in \mathcal{A}} \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{err^2\left(t(a), t_{pred}(a)\right)}{2\sigma^2}} =$$

$$= \underset{t_a(V), t_b(V)}{\arg\max} \sum_{a \in \mathcal{A}} \ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{err^2\left(t(a), t_{pred}(a)\right)}{2\sigma^2}.$$

Členy $\ln\left(\frac{1}{\sigma\sqrt{2\pi}}\right)$ jsou nezávislé na $t_a(V), t_b(V)$, můžeme je tedy odstranit, stejně tak koeficient $\frac{1}{2\sigma^2}$ je shodný pro všechny členy součtu. Po zjednodušení tedy

$$= \underset{t_a(V), t_b(V)}{\arg\max} \sum_{a \in \mathcal{A}} -\frac{err^2\left(t\,(a), t_{pred}\,(a)\right)}{2\sigma^2} =$$

[6] logaritmus je ryze rostoucí funkce a hustota rozložení normálního rozdělení je kladná pro všechna reálná čísla, nemá to tedy vliv na hledání arg max

$$= \underset{t_a(V), t_b(V)}{\arg\min} \sum_{a \in \mathcal{A}} \left(\ln\left(t_{pred}\left(a\right)\right) - \ln\left(t\left(a\right)\right) \right)^2 .$$

Nejlepší aproximaci skutečného T tedy bude mít model minimalizující

$$loss := \sum_{a \in \mathcal{A}} \left(\ln\left(t_{pred}\left(a\right)\right) - \ln\left(t\left(a\right)\right) \right)^2 ,$$

což budeme označovat jako ztrátovou funkci a její hodnotu (resp. průměr na jednu akci) používat k hodnocení úspěšnosti modelů. Pro zjednodušení budeme jako $loss\left(t_{pred}\left(a\right), t\left(a\right)\right)$ označovat $\left(\ln\left(t_{pred}\left(a\right)\right) - \ln\left(t\left(a\right)\right) \right)^2$, tedy člen sumy pro akci a.

3.3 Detekce a zpracování outlierů

Výjimečná data, ať už zahlášenou hodnotou (v našem případě dvojice „čas-počet kusů“ u akcí) či popisem (například unikátní typ operace) obecně snižují úspěšnost algoritmů strojového učení. Tato data můžou být způsobena chybou či výjimečnou situací při jejich získávání (například speciální požadavek zákazníka na jednorázovou změnu v určité operaci). Příklady outlierů mezi zahlášenými hodnotami lze vidět na obrázku 3.1.

Pro zlepšení výsledků použitých metod tedy nejprve detekujeme outliery a zpracujeme je. Za jejich trénovací množiny pak budeme považovat data po těchto úpravách.

3.3.1 Problém vzácně se vyskytujících atributů

Nejprve se zaměříme na velmi vzácné atributy v informacích o jednotlivých operacích a produktech. Typickým příkladem jsou kategorická data, případně znaky řetězců, které jsou rovněž druhem kategorií. Řada kategorií je přítomna pouze pro jednotky akcí, tudíž jakékoli predikce pro takové kategorie nejsou relevantní. Nadbytečné kategorie navíc zesložiťují modely, čímž zpomalují trénování a zvyšují výpočetní nároky.

Máme dvě hlavní možnosti, jak tyto problémy řešit, případně lze použít jejich kombinaci:

- nahrazení vzácných kategorií jednou novou kategorií,
- odstranění dat s danými kategoriemi.

Výběr mezi těmito možnostmi by měl být proveden dle charakteristiky dané informace. Pokud řetězec obsahuje vzácný znak, není vhodné odpovídající datový záznam zcela odstranit, ani vyjmout pouze daný znak, neboť tím bychom změnili strukturu tohoto řetězce. V tomto případě je tedy vhodné nahradit všechny vzácné znaky hodnotou „neznámý“ – pak jednotlivé učící algoritmy mohou využít fakt, že na daném místě nějaký netypický znak byl. Podobnou argumentaci lze využít u informací složených z množiny kategorií.

Na druhou stranu, velmi vzácné kategorie v některém kategorickém atributu nemusí být vhodné slučovat, neboť nejspíše odpovídají různorodým pracovním

postupům. Jelikož je celá informace tvořena pouze jedinou kategorií, neobsahuje přímo další hodnoty pro vytvoření „kontextu“ jako v předchozím případě. Mohlo by tedy být vhodnější data s takovými kategoriemi zcela smazat.

Data ze vzácných kategorií se však vyskytují i ve validačních množinách, nějakou predikci je pro ně tedy třeba udělat. Jak uvidíme v kapitole 7, slučování vzácných kategorií nemá na výsledky modelů založených na lineární regresi velký vliv. Modely založené na neuronové síti se užitečnost jednotlivých informací naučí samy. Variantu mazání dat se vzácnými kategoriemi tudíž nebudeme provádět.

3.3.2 Problém nestandardních zahlášených hodnot

Ve vstupních datech se nachází malé množství atypických akcí či chyb. Takovéto akce mohou být velkým problémem především pro algoritmy, které jsou na outliery citlivé. To lze pozorovat například u operace typu 20 na obrázku 3.2, kde 4 akce se zahlášeným časem v řádu 10^4 minut pro standardní lineární regresi způsobí posunutí predikovaného přípravného času a téměř nulový korelační koeficient.

Existuje velké množství algoritmů, které se liší především typy outlierů, které jsou schopny detekovat. Podrobný rozbor můžeme nalézt v článku Malik a kol. (2014). Základními variantami jsou tzv. bodový outlier (zahlášená akce izolovaná od ostatních, příklad lze nalézt na obrázku 3.5) a kolektivní outliery (několik navzájem blízkých akcí, které ovšem vybočují od ostatních akcí pro stejnou operaci, viz obrázek 3.6).

V ideálním případě outliery skutečně vznikly chybou, tudíž bychom je měli z dat odstranit. Nicméně některé z nich mohou mít skutečný důvod, predikce by je tedy měla brát v úvahu. Naopak některá data, která nejsou outliery, mohou být ve skutečnosti chybná. Problémem je, že nemáme k dispozici informaci o tom, zda je daná akce chybná či nikoli. Vyzkoušíme tedy více metod a jejich parametrů a krosvalidací určíme vhodné hodnoty pro jednotlivé estimátory.

Implementované metody pro detekci budou založeny na těchto myšlenkách:

- výpočet Z-skóre, kde předpokládáme chování původních dat dle normálního rozdělení,
- výběr dle „izolovanosti“ jednotlivých akcí – využijeme myšlenku metody DBSCAN.

Vzhledem k tomu, že jednotlivé operace z $\mathcal{V}$ mají příliš nízký počet zahlášených akcí pro efektivní detekci, budeme ji provádět pro celé skupiny operací sdružené dle jejich typu. Počet akcí pro jednotlivé typy a jejich rozptyl se však výrazně liší, není proto vhodné zvolit stejné parametry pro všechny typy. Namísto toho vyjdeme z předpokladu, že frekvence chyb pro jednotlivé typy operací by měly být srovnatelné. Jako parametr tedy zvolíme očekávanou pravděpodobnost p_o, že náhodná akce bude outlier. Poté pro každý typ operace nalezneme takové parametry specifické pro danou detekční metodu, aby byl poměr vybraných outlierů vzhledem ke všem akcím dané operace blízký p_o.

Detekce pomocí Z-skóre

Základem této metody je předpoklad, že akce z $\mathcal{A}$ pocházejí z normálního rozdělení s určitými parametry. Tento předpoklad je v našem případě zjevně neprav-

divý, ovšem můžeme je pro tuto metodu upravit tak, aby detekce byla ve shodě s hodnotící funkcí ze sekce 3.2.

K výpočtu Z-skóre chceme nalézt takové parametry normálního rozdělení, které odpovídají nejvěrohodnějšímu modelu vzhledem k datům, tj. maximalizujeme

$$P(\mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})|X).$$

Obvykle se při výpočtu Z-skóre předpokládá nezávislost v jednotlivých rozměrech, tj. že kovarianční matice odhadovaného normálního rozdělení $\boldsymbol{\Sigma}$ je diagonální. V našem případě však toto zjednodušení udělat nemůžeme – hledáme lineární závislost mezi zahlášeným časem a počtem kusů. Kromě toho očekáváme, že rozptyl zahlášených časů lineárně roste se skutečným výrobním časem, neboť $err(x, y) = \ln(x) - \ln(y) = \ln(2x) - \ln(2y) = err(2x, 2y)$. Proto je nejspíše vhodnější hledat outliery pro zlogaritmovanou hodnotu zahlášeného času, aby očekávaná hustota akcí byla zachována. Tato úprava však nemusí být vždy ideální, neboť předpokládaná lineární závislost $n(a)$ a $t(a)$ není lineární pro $\ln\Big(t(a)\Big)$ – v kapitole 7 srovnáme účinnost použití logaritmu pro jednotlivé modely.

Řešení této maximalizační úlohy je dobře známé (viz Malik a kol., 2014). V případě použití logaritmu jsou všechny zahlášené časy $t(a)$ změněny ještě před hledáním řešení, v řešení tedy stačí výskyty $t(a)$ nahradit $\ln\Big(t(a)\Big)$.

$$\boldsymbol{\mu} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Big(n(a), t(a)\Big),$$

$$\boldsymbol{\Sigma} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \Big(\Big(n(a), t(a)\Big) - \boldsymbol{\mu}\Big) \Big(\Big(n(a), t(a)\Big) - \boldsymbol{\mu}\Big)^T.$$

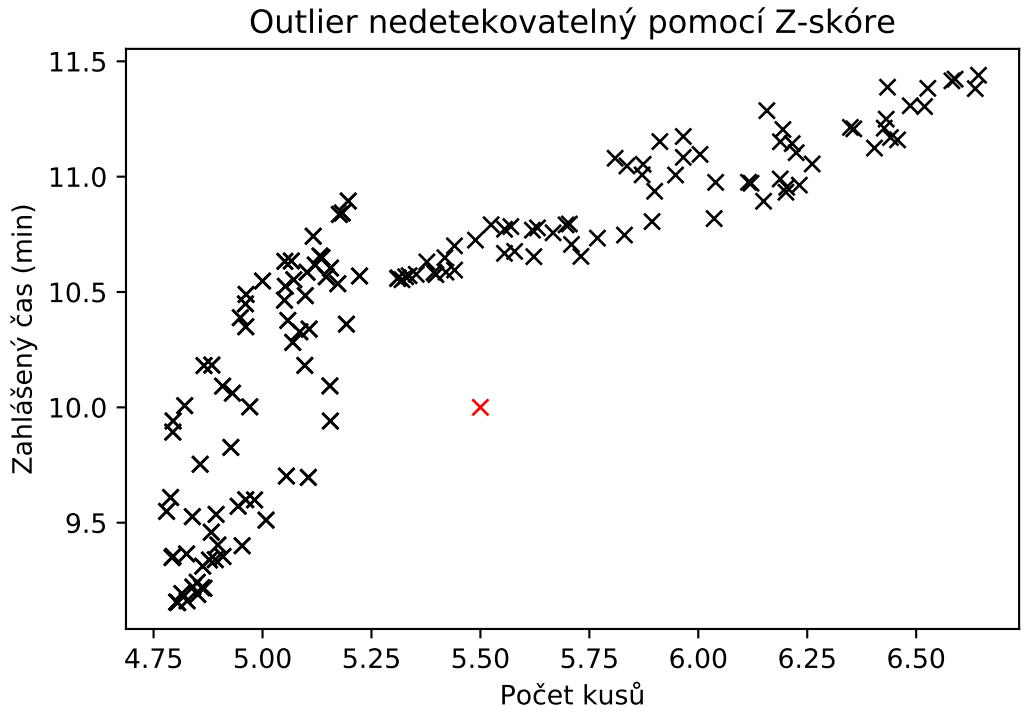
Jako Z-skóre akce a se poté označuje počet standardních odchylek dvojice $\Big(n(a), t(a)\Big)$ vzhledem ke spočtené $\mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Akce s vyšším Z-skóre by tudíž měly odpovídat méně pravděpodobným situacím. Budeme-li následně chtít odebrat například 10 největších outlierů, stačí si všechny akce seřadit sestupně dle Z-skóre a vynechat prvních deset akcí.

Problémem pro tuto metodu je detekce takových akcí a, které jsou sice vzdálené od ostatních akcí, nicméně nejsou příliš daleko od spočteného $\boldsymbol{\mu}$. Příklad takové akce a můžeme nalézt na obrázku 3.5. Vzhledem k tomu, že akce a by však neměla být daleko ani od skutečného výrobního času $T\Big(V(a)\Big)$, ponechání takového outlieru v trénovací množině by nemělo způsobit výraznější problém.

V reálných datech lze spíše předpokládat výskyt akcí nesprávně označených jako outliery – typickým příkladem je typ operace, který se téměř vždy provádí jen na jednom kusu produktu. Pro takový případ vyjde střední hodnota počtu kusů blízko jedné a standardní odchylka v této ose bude malá, čímž Z-skóre akcí s více kusy bude vždy vysoké bez ohledu na zahlášený čas.

Detekce pomocí hustoty akcí

Největší nevýhodou Z-skóre je předpoklad normálního rozdělení zahlášených akcí, což nemusí vždy odpovídat skutečnosti, jak bylo popsáno na konci předchozí sekce 3.3.2. Chtěli bychom tedy nalézt alternativní způsob detekce outlierů minimalizující počet předpokladů o rozdělení dat. Jednou z možností je algoritmus



Obrázek 3.5: Příklad bodového outlieru, který nebude detekován pomocí Z-skóre

DBSCAN (zkratka z Density Based Spatial Clustering of Applications with Noise (Ester a kol., 1996)), který je založený na konstrukci clusterů dle „blízkosti“ jednotlivých datových bodů.

Prvním krokem algoritmu je určení minimálního počtu $n_{min} \in \mathbb{N}$ blízkých bodů nutných pro označení bodu jako jádrový, definování vzdálenosti mezi jednotlivými body $dist(a, b)$ a volba vzdálenostního limitu $\varepsilon > 0$, do kterého jsou dva body považovány za blízké. Jako $dist(a, b)$ se typicky využívá eukleidovská vzdálenost. V takovém případě je vhodné vstupní data přeškálovat tak, aby byly rozsahy v každé ose podobné, v našem případě počty kusů a zahlášené časy (obdobně jako u Z-skóre s možností předchozí logaritmizace zahlášeného času). Následně pro každý bod a nalezneme všechny body b ze vstupní množiny, pro které $dist(a, b) \leq \varepsilon$. Je-li počet takových bodů (včetně a) alespoň n_{min}, je a označen jako jádrový. Po nalezení všech jádrových bodů označíme jako okrajové všechny body, které nebyly označeny jako jádrové, ovšem jsou zároveň ve vzdálenosti nejvýše ε od nějakého jádrového bodu. Body, které nebyly označeny jako jádrové ani jako okrajové budeme považovat za outliery.

Jelikož je naším cílem využít metodu DBSCAN pro detekci outlierů, tato sekvence kroků je pro nás dostačující. DBSCAN však nijak negarantuje počet bodů, které s danými parametry označí jako outliery. Zbývá tedy vyřešit tento problém, k čemuž můžeme využít úpravy jeho parametrů:

- $dist$: Změny tohoto parametru nejsou příliš vhodné, neboť ve více rozměrech není zřejmé jak ji upravit při zachování původní myšlenky algoritmu
- n_{min}: Tento parametr lze interpretovat jako hranice pro hustotu bodů aby nebyly označeny jako outliery, omezující především malé clustery. Výhodou je neklesající závislost počtu outlierů na n_{min} – zvýšení n_{min} může pouze

způsobit neoznačení nějakého bodu jako jádrový, čímž v druhé části algoritmu může nějaký původně okrajový bod přijít o všechny jádrové body do vzdálenosti ε a stát se tak outlierem. Lze tedy využít binární vyhledávání pro určení ideálního n_{min}. Nevýhodou je omezení na přirozená čísla, tj. počty outlierů dvou sousedních hodnot n_{min} mohou být v obou případech daleko od požadovaného počtu.

- ε: Tento parametr omezuje vzdálenost, se kterou jsou ještě body považovány za blízké, jeho zvyšování tedy označuje za outliery spíše oblasti s menší hustotou bodů. Obdobně jako u n_{min} je počet outlierů nerostoucí vzhledem k ε – snížení ε může rovněž způsobit neoznačení bodu jako jádrový, původně okrajový bod se navíc může stát outlierem i beze změny množiny jádrových bodů, čímž budou skokové změny v počtu outlierů menší. Narozdíl od n_{min} může být ε libovolné kladné reálné číslo, binární vyhledávání ε bude tedy typicky výrazně úspěšnější s přiblížením se k požadovanému počtu outlierů.

Nevýhodou této metody je pomalejší běh vzhledem k nutnosti DBSCAN opakovat s různými parametry. Rovněž není schopna detekovat outliery v případě, že je tvoří několik blízkých akcí nehledě na vzdálenost od zbytku akcí, viz obrázek 3.6. Taková situace může občas pro reálná data nastat, například když nějaký pracovník celý den zadává zahlášené časy se špatnou časovou jednotkou. Narozdíl od Z-skóre může však mít takový nedetekovaný outlier velký vliv na predikci některých modelů kvůli vysoké hodnotě *err* vzhledem k skutečnému výrobnímu času.

Pro úplnost zmiňme zbývající část algoritmu DBSCAN, tedy samotnou clusterizaci. Ta se provádí následujícími třemi kroky:

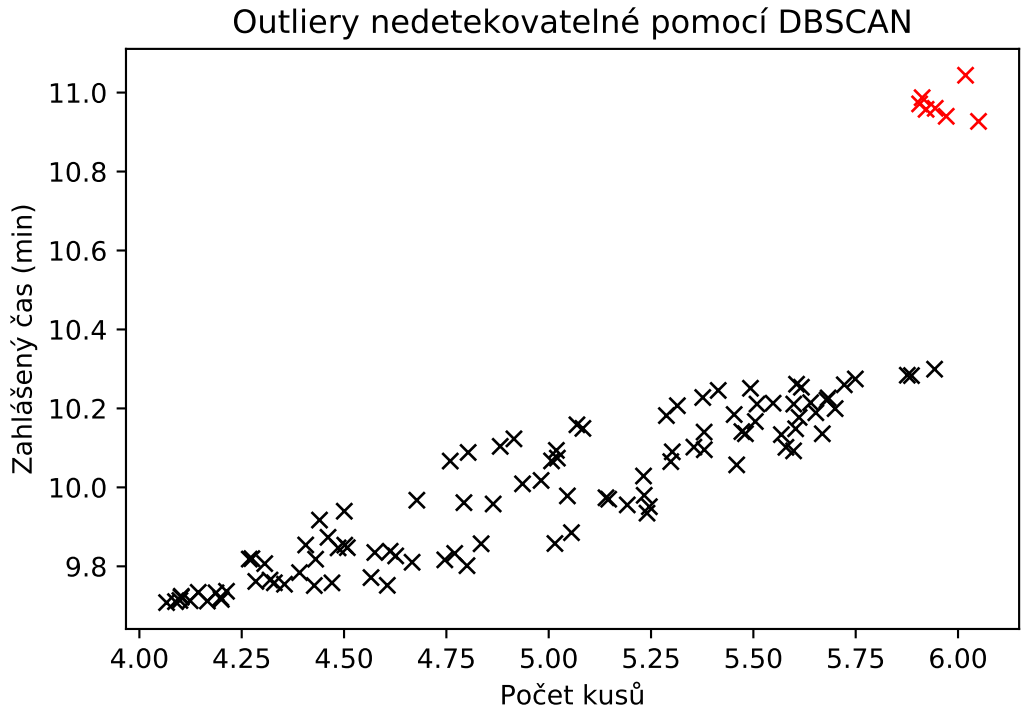
- Zvolíme libovolný jádrový bod, který ještě není přiřazený do žádného clusteru, a vytvořme pro něj nový cluster C.
- Do C přidáme všechny body, které jsou ve vzdálenosti nejvýše ε od nějakého jádrového bodu v C a ještě nebyly do žádného clusteru přiřazeny.
- Předchozí bod opakujeme, dokud jeho aplikování přidá do C alespoň jeden nový bod.

Tyto kroky budeme provádět do té doby, dokud zbývá nějaký jádrový bod nepřiřazený do žádného clusteru.

Povšimněme si, že všechny jádrové body, které jsou vzdálené nejvýše ε, budou vždy přiřazeny do společného clusteru. Outliery naopak nebudou nikdy přiřazeny do žádného clusteru. Okrajový bod může v některých situacích být přiřazen do různých clusterů v závislosti na pořadí zpracování jádrových bodů, je-li vzdálen nejvýše ε od dvou různých jádrových bodů které nejsou vzájemně propojeny. V takovém případě ho algoritmus DBSCAN přiřadí do prvního vhodného clusteru dle zvoleného pořadí zpracování jádrových bodů.

3.4 Vytvoření testovacích dat

Data (konkrétně množinu $\mathcal{A}$) je třeba rozdělit na trénovací a testovací množinu, přičemž musíme předem určit, jaká část bude určena pro testování. V této



Obrázek 3.6: Příklad kolektivních outlierů, které nebudou detekovány na základě hustoty akcí

práci budeme pro testování používat $c = \frac{|\mathcal{A}|}{10}$ dat. To by vzhledem k velikosti $\mathcal{A}$ měla být statisticky dostatečně významná množina a vzhledem ke komplexnosti atributů chceme maximalizovat množství trénovacích dat.

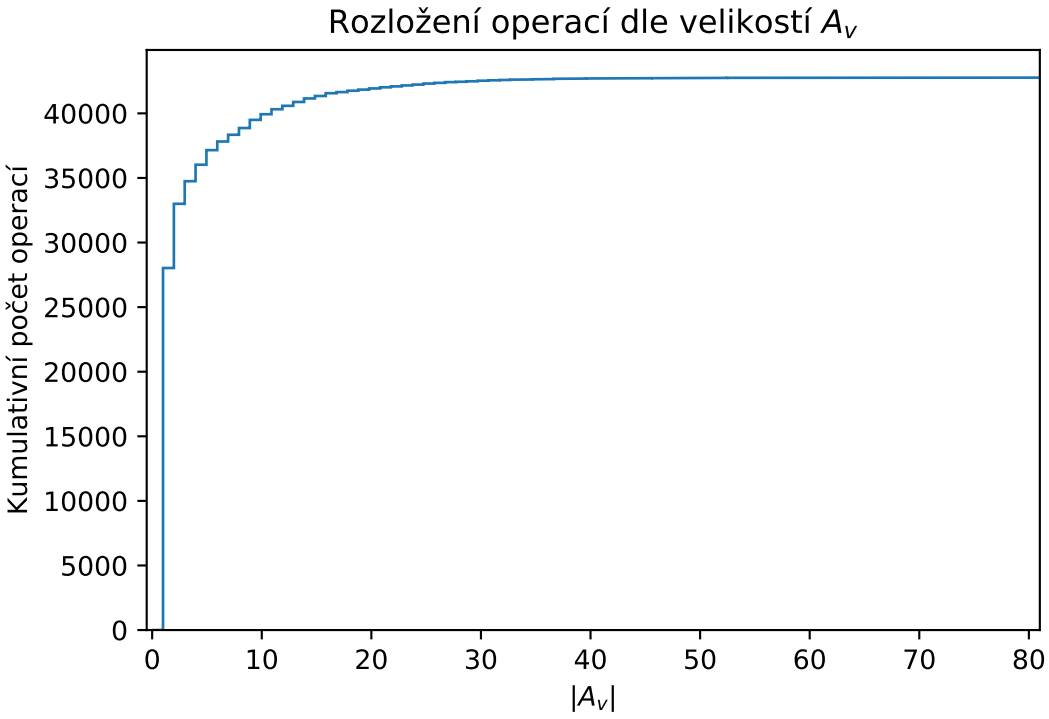
Výběr testovací množiny můžeme provést různými způsoby. Jednotlivé způsoby výběru budeme souhrnně označovat C_i, jimi vybrané trénovací množiny C_i^t a testovací množiny C_i^f, kde i je konkrétní typ výběru.

Vstupní data bohužel nejsou vzájemně nezávislá. Pokud bychom tedy do trénovací množiny vybírali náhodnou c-prvkovou podmnožinu $\mathcal{A}$ (označme takový způsob výběru C_1), akce odpovídající stejným operacím by byly rozděleny mezi trénovací i testovací množinu. Modely z nich při predikci mohou vycházet, čímž bychom příliš neověřili schopnost jeho generalizace. Na takové testovací množině by tedy měly dosahovat lepších výsledků, než jaké bychom měli dostat pro zcela nezávislá data. Distribuce akcí v C_1^f nicméně odpovídá očekávané distribuci nově prováděných akcí ve zdrojové firmě, tudíž by měl výsledek odpovídat očekávané přesnosti predikce těchto akcí.

Popsaná selekce C_1 není vhodná pro ohodnocení výkonnosti modelů na nových operacích. Pokud však do testovací množiny přesuneme vždy všechny akce $\mathcal{A}_V$ spojené s vybranou operací V, není pro takovou operaci při trénování známá žádná akce. Predikce tak musí být odvozena pouze z podobnosti popisu V s ostatními operacemi, čímž dosáhneme dobré úrovně nezávislosti. Selekci C_2 tedy vytvoříme tak, že náhodně uspořádáme všechny operace $(V_1, V_2, \ldots, V_{|\mathcal{V}|})$ a použijeme

$$C_2^f = \bigcup_{i \leq k} \mathcal{A}_{V_i},$$

kde k je nejmenší číslo, pro které dosáhne $\left|C_2^f\right| \geq c$.



Obrázek 3.7: Kumulativní histogram rozdělení operací dle velikosti jejich A_V.

Výsledky na této množině by měly odpovídat úspěšnosti predikce pro nové operace a produkty. Lze tak očekávat, že budou obecně horší než pro C_1.

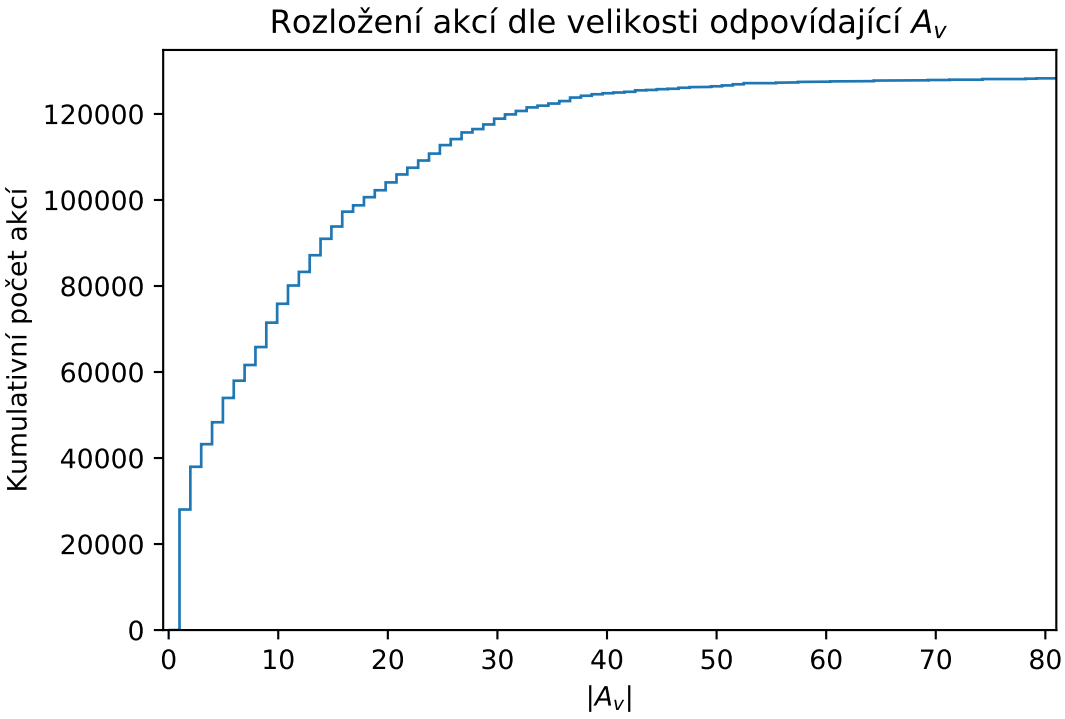
Poslední variantou selekce C_3 je postup podobný expertnímu – testovací množina bude vybírána pouze z „často“ prováděných akcí. Vytvoříme ji tak, že z některých tříd akcí splňujících $|\mathcal{A}_V| \geq k$ pro nějaké $k > 0$ náhodně vybereme (dolu zaokrouhlenou) polovinu akcí do testovací množiny. Predikci tedy budeme provádět pouze pro často prováděné operace. Obdobně jako v předchozím případě budeme volit náhodně mezi těmito třídami až do naplnění C_3^f.

V našem případě budeme používat $k = 9$, tj. v trénovací množině C_3^t bude vždy alespoň pět akcí libovolné operace z C_3^f. Tato hodnota byla zvolena jako dobrý kompromis mezi minimálním počtem akcí vybraných operací a celkovým počtem vhodných operací – pro 3904 operací bylo provedeno alespoň 9 akcí, přičemž celkem je takových akcí 62651, tj. přibližně polovina dat. Tento počet s rostoucím k prudce klesá, například pro $k = 10$ je to již jen 3274 operací a 56981 akcí. Detailní rozložení lze nalézt na obrázcích 3.7 a 3.8.

Tato testovací množina je zjevně výrazně provázaná s trénovací, je vhodná pro vyhodnocení predikcí pro často prováděné operace. Můžeme ji tudíž využít především pro vyhodnocení jednoduchých modelů a jejich schopnosti zastoupit technologa výroby, které ale současně nedokážou účinně predikovat pro zbylé varianty.

3.4.1 Vyhodnocování výsledků

Pro všechny modely budeme používat stejné množiny $C_{1,2,3}^t$ a $C_{1,2,3}^f$, abychom je mohli mezi sebou porovnat. Jelikož však chceme porovnat větší množství modelů, jejichž parametry budou motivovány výsledky na těchto testovacích mno-



Obrázek 3.8: Kumulativní histogram rozdělení akcí dle velikosti A_V, do které daná akce přísluší.

žinách, s rostoucím počtem parametrů by docházelo k zahrnování testovacích množin do učení a tím i k nadhodnocování výsledků.

K zachování korektnosti tak nesmíme testovací množiny použít jinde než v závěrečném srovnání. Na druhou stranu však můžeme z trénovacích množin oddělit validační množiny a odhadovat výsledky pro testovací množiny na základě těchto pomocných. Při velkém množství dat by bylo možné z každé trénovací množiny vyčlenit validační množinu pevnou, nicméně vzhledem k počtu informací o každé operaci bude v našem případě vhodnější využít krosvalidaci:

- Všechna data z C_i^t rozdělíme na $d = 5$ dílů dle odpovídajících pravidel C_i.
- Každý z těchto dílů použijeme jednou jako validační množinu, jinak bude vždy součástí trénovací množiny. Získáme tedy d učebních sad.
- Pro zvolené kombinace parametrů zvalidujeme trénovaný model vždy na všech d sadách, jeho finální parametry pak vybereme dle výsledků.
- Finální model natrénujeme na celé C_i^t a vyhodnotíme na C_i^f.

4. Modely založené na lineární regresi (LR)

Standardní lineární regrese je jedním z nejjednodušších modelů, který lze pro řešení tohoto problému využít. Vzhledem k tomu, že výrobní časy jsou z definice lineární funkcí počtu kusů, je klasická LR vhodným základem.

V metodách diskutovaných v této kapitole nebudeme využívat dodatečné informace o jednotlivých produktech a operacích kromě typu operace jako zálohy, dostaneme-li požadavek na predikci pro zcela neznámou operaci. Predikce tak pro operaci $V \in \mathcal{V}$ bude vždy založena pouze na akcích $a \in \mathcal{A}_V$ dostupných během trénování, s možností využití akcí všech operací stejného typu jako V rovněž dostupných během trénování. Jak již bylo zmíněno v úvodu, tímto postupem získáme velmi nízkou složitost modelu, přijdeme však o schopnost predikovat výrobní časy pro vzácné či nové operace. Pro učení rovněž nebudeme vždy přesně využívat chybovou funkci *err* definovanou v kapitole 3. Namísto toho použijeme její aproximace, jelikož není vhodná pro každý typ LR.

4.1 Problémy specifické pro tuto metodu

4.1.1 Neplatné výsledky regrese

Vstupní data nám nedávají žádné garance, jakým způsobem vyjde hledaná LR – víme pouze, že všechny akce mají kladný výrobní čas i počet vyrobených kusů. Je však možné vygenerovat takové akce, aby optimální přímka pro danou operaci V protínala osu y v záporných hodnotách (tj. predikovala záporný přípravný čas $t_b(V)$), či měla zápornou směrnici (tj. predikovala záporný jednicový čas $t_a(V)$).

Při takové situaci musíme pro smysluplnou predikci $T(V)$ použít záložní řešení, které pro libovolné V vytvoří nezáporná $t_b(V), t_a(V)$. Robustním řešením je zafixování $t_b(V) = 0$ a optimalizace pouze t_a – jelikož všechny zahlášené akce mají kladný počet kusů i čas, odpovídají bodům ostře v prvním kvadrantu. Každá smysluplná varianta LR se zafixovaným průchodem počátkem zvolí hodnotu pro směrnici $t_a(V)$ z intervalu omezeného minimem a maximem $\frac{t(a)}{n(a)}$ akcí a z A_V, bude tedy jistě kladná. Tímto způsobem bohužel ignorujeme čas nutný k přípravě stroje, nicméně tuto hodnotu nejsme schopni vzhledem k výsledkům regrese smysluplně odhadnout.

Alternativní možností je „simulovat“ přípravný čas jako výrobu $r \in \mathbb{R}_0^+$ kusů. Všem akcím zvýšíme počet vyrobených kusů o r a použijeme predikci dle předchozího odstavce. Jako $t_b(V)$ pak použijeme $r \cdot t_a(V)$. Otázkou však zůstává, jakým způsobem nastavit hodnotu r – přípravný čas by měl pro jednotlivé typy operací přibližně odpovídat konstantnímu počtu vyrobených kusů, tudíž můžeme pro zjednodušení provést následující kroky:

- Vytvoříme regresor nad všemi akcemi z trénovacími množiny s daným typem operace a získáme jím predikované t_b a t_a.
- V případě, že t_b či t_a nevyjde kladné (pravděpodobnost takové situace je menší, neboť využíváme velké množství akcí), použijme $r = 0$.

- Jinak spočteme očekávaný počet „virtuálních“ kusů jako $r = \frac{t_b}{t_a}$, neboli počet „virtuálních“ kusů, které bychom museli odebrat aby predikce splnila $t_a \cdot (-r) + t_b = 0$.

4.1.2 Predikce pro neznámé operace

Jelikož nedokážeme provést predikci pro žádnou operaci, pro kterou jsme neměli odpovídající akce v trénovací množině, nelze pro $T(V)$ použít predikci přímočaře. Tyto modely tedy dokážeme korektně otestovat pouze na C_3, která garantuje pro každou akci $a \in C_3^f$ existenci akcí $a' \in C_3^t$ s $V(a') = V(a)$. Pro zbylé testovací množiny musíme pro neznámé operace vytvořit záložní predikci.

Vytvoříme tedy několik predikcí výrobních časů $T_i \in \left(\mathbb{R}_0^+\right)^3$ a pro každou operaci V, pro kterou nejsme schopni provést predikci pomocí regresoru přímo, použijeme odhad $T(V) = T_i$ pro nějaké i. Vzhledem k tomu, že výrobní čas je obvykle značně závislý na typu prováděné operace, T_i vygenerujeme použitím zvolené LR na všechna trénovací data s typem operace i. Jelikož typů operací očekáváme nejvýše několik desítek, můžeme předpokládat, že v trénovací množině budou všechny typy operací dostatečně zastoupeny pro provedení predikce. Jako záložní řešení můžeme rovněž provést predikci na všech trénovacích datech a v případě výskytu zcela neznámého typu operace ve validační či testovací množině pak použijeme tuto predikci. Taková predikce nejspíše nebude příliš dobrá, nicméně vzhledem k očekávanému počtu akcí, pro které ji bude nutné použít, je případné zvýšení chyby zanedbatelné.

4.1.3 Chyby v datech

V datech vyskytuje malé množství výrazných chyb v atributech a přesnosti zahlášených časů. Vzhledem k tomu, že se tak rozptyl *err* zcela neřídí normálním rozdělením, může dávat standardní LR a její modifikace špatné výsledky. Musíme tedy taková data buďto odfiltrovat (například pomocí algoritmů popsaných v sekci 3.3) nebo použít takové varianty LR, které budou vůči těmto problémům robustní.

4.2 Popis LR modelů

Vzhledem k tomu, že u všech akcí nyní budeme uvažovat jako nezávislou proměnnou pouze počet vyrobených kusů, u všech metod LR budeme využívat jejich variantu s jednou nezávislou proměnnou, kterou je v našem případě počet kusů jednotlivých zahlášených akcí. Tvrzení o existencích a výpočtu řešení pro standardní a váženou verzi LR vycházejí z knihy „Linear Regression Analysis“ (Seber, 2003).

4.2.1 Standardní LR

Klasická varianta LR hledá přímku $t_{pred}(x) = t_a x + t_b$ takovou, která pro daná data $((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$ minimalizuje

$$\sum_{i=1}^{n} \left(t_{pred}\left(x_i\right) - y_i\right)^2.$$

Označme

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ & \vdots \\ 1 & x_n \end{pmatrix}, Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Pokud jsou sloupce X lineárně nezávislé (tj. mají-li alespoň 2 akce odlišný počet kusů), je hledaná přímka jednoznačně určena vztahem

$$\begin{pmatrix} t_b \\ t_a \end{pmatrix} = \left(X^T X\right)^{-1} X^T Y.$$

V případě, že je pro akce A_V dané operace V matice $X^T X$ invertibilní a vypočtené t_b, t_a jsou nezáporné, můžeme tento výsledek rovnou použít jako predikci $T(V)$. V opačném případě budeme postupovat dle diskuze v sekci 4.1.1 – zafixujeme $t_b\,(V) = 0$ a případně virtuálně zvýšíme počet kusů akcí v A_V o r. Při zafixování $t_b\,(V) = 0$ se změní výpočet optimálního t_a:

$$\min_{t_a} \sum_{i=1}^{n} (t_{pred}(x_i) - y_i)^2 = \min_{t_a} \sum_{i=1}^{n} (t_a \cdot x_i - y_i)^2 \iff$$
$$\iff \sum_{i=1}^{n} (t_a \cdot x_i - y_i) y_i = 0 \iff t_a = \frac{X'^T Y}{|X'|^2},$$

kde $X' = (x_1, x_2, \ldots, x_n)^T$.

4.2.2 Vylepšení standardní LR

Hlavní nevýhodou klasické varianty LR je fakt, že je předpokládána tzv. unimodularita dat, tedy že se zahlášený čas akcí chová dle

$$y_i = t_a x_i + t_b + \varepsilon,$$

kde ε je náhodná veličina s normálním rozdělením $\mathbb{N}(0, \sigma^2)$ pro nějaké $\sigma > 0$. Šum tudíž předpokládáme nezávislý na počtu vyrobených kusů i na zahlášeném čase. To ale neodpovídá skutečnosti, jak víme z kapitoly 3. Rovněž nemůžeme zahlášené časy přímo zlogaritmovat, čímž by sice chybová funkce v standardní LR již *err* odpovídala, nicméně lineární predikce v takto modifikovaných datech by nebyla lineární v původních datech.

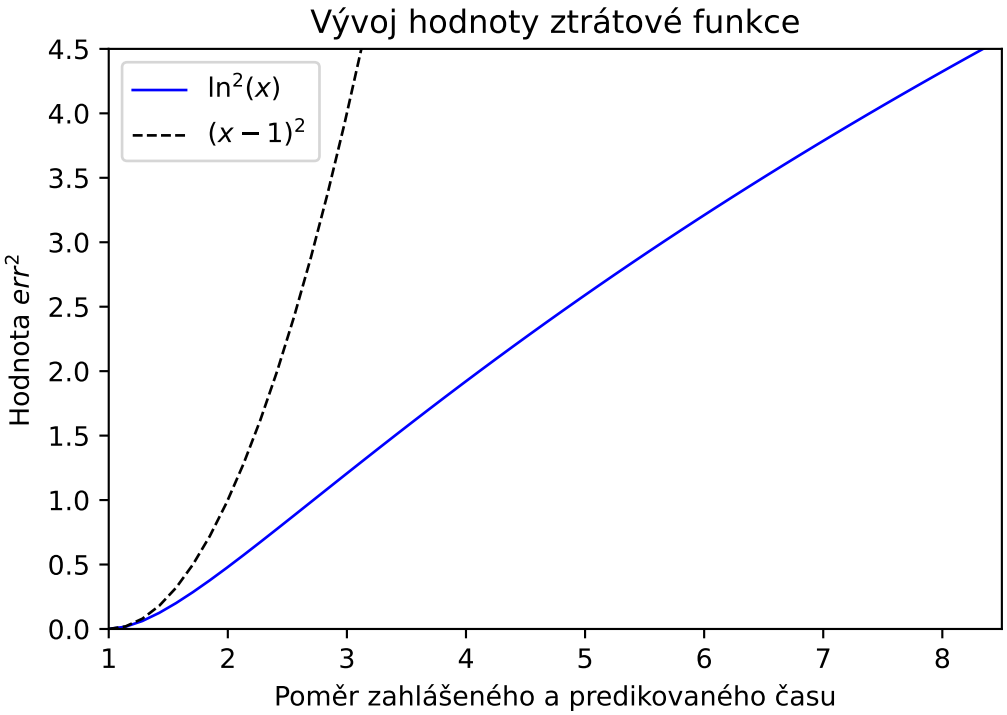
Máme více možností, jak použitou chybovou funkci pro trénování modelů upravit, aby byla blíže *err*. Především pak varianty

$$(1 + \varepsilon) y_i = t_a x_i + t_b,$$

$$y_i = (t_a x_i + t_b)(1 + \varepsilon),$$

$$y_i = (t_a x_i + t_b) \mathrm{e}^{\varepsilon},$$

kdy v prvním případě modelujeme chybu jako relativní vzhledem k zahlášenému času, ve druhém pak ke skutečnému času operace. Pro jednoduchost opět předpokládejme $\varepsilon \in \mathbb{N}(0, \sigma^2)$, ve druhém případě oříznuté na $[-1, \infty)$. Výhodou těchto



Obrázek 4.1: Porovnání ztrátové funkce pro relativní a log-relativní chybu predikce.

variant je podobnost s *loss* pro akce s $t(a)$ blízkým $t_{pred}\left(V(a)\right)$, jak můžeme vidět na obrázku 4.1. Třetí varianta odpovídá skutečnému rozdělení err při použití $err(1, e) = 1$, neboť

$$\ln(y_i) = \ln(t_a x_i + t_b) + \varepsilon,$$
$$\varepsilon = \ln(y_i) - \ln(t_a x_i + t_b).$$

nicméně jak víme z důsledku věty 3 nemá hodnota $err(1, e)$ tento koeficient vliv na hledání optima. Jelikož je však ε v exponentu, dosazením $\mathbb{N}(0, \sigma^2)$ za ε a pokusem o maximalizaci pomocí derivace již nedostaneme soustavu lineárních rovnic a nelze tak využít standardní algebraické metody ostatních LR.

Standardní odchylka založena na zahlášeném času – vážená LR

Nejprve rozebereme vylepšení modelu použitím $(1 + \varepsilon)y_i = t_a x_i + t_b$.

Problémem modelování chyby vzhledem k zahlášenému času je to, že ε nikdy nevyjde menší než -1, tudíž jeho skutečné rozdělení zcela neodpovídá normálnímu. Zejména v případě velmi nízkého zahlášeného času by navíc mělo ε vyjít velmi vysoké, tj. vysoce nepravděpodobné. Na druhou stranu, výhodou je jednoduchá implementace, neboť je tento model ekvivalentní $y_i = t_a x_i + t_b - y_i \cdot \varepsilon$, kde y_i je známé. Maximálně věrohodný odhad tedy dostaneme metodou nejmenších čtverců obdobně jako u standardní LR, ovšem jeho váženou variantou kvůli

koeficientům y_i:

$$\max_{t_a,t_b} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{t_a x_i + t_b - y_i}{2\sigma y_i}\right)^2} =$$
$$= \max_{t_a,t_b} \sum_{i=1}^{n} - \left(\frac{t_a x_i + t_b - y_i}{y_i}\right)^2 =$$
$$= \min_{t_a,t_b} \sum_{i=1}^{n} \frac{1}{y_i^2} \left(t_a x_i + t_b - y_i\right)^2$$

což přesně odpovídá minimalizaci vážené střední kvadratické chyby s váhami $\frac{1}{y_i^2}$. Tu lze řešit stejným způsobem jako neváženou verzi (viz Seber, 2003) při použití

$$X = \begin{pmatrix} 1/y_1 & x_1/y_1 \\ 1/y_2 & x_2/y_2 \\ & \vdots \\ 1/y_n & x_n/y_n \end{pmatrix}, Y = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Byť tento model bere v úvahu odchylku závislou na zahlášeném čase, jeho výsledky na validačních množinách při krosvalidaci nebyly výrazně lepší. To je dáno zejména velkým nadhodnocováním nízkých zahlášených hodnot – přestože se v reálné situaci pravděpodobnost zahlášení 10 a 20 minut namísto správných 15 minut výrazně neliší, tento model nastaví první variantě dvojnásobnou váhu. Každá výrazněji časově podhodnocená akce si tedy vynucuje blízko vedoucí výslednou regresi.

Standardní odchylka založena na skutečném času – iterativní LR

Předchozí problém by se neměl vyskytovat v případě $y_i = (t_a x_i + t_b)(1 + \varepsilon)$. U této varianty je rozptyl náhodné proměnné lineární se skutečným časem operace, tudíž například v příkladu z předchozího odstavce nastaví akcím s 10 i s 20 minutami stejnou váhu. Vzhledem k tomu, že je obvykle mírně pravděpodobnější skutečný čas překročit než skončit dříve (obdobně jako u předchozí situace je ε omezeno zdola -1), bude tato metoda skutečný čas nejspíše mírně nadhodnocovat.

Hlavní nevýhodou je však nemožnost algebraického nalezení optima:

$$\max_{t_a,t_b} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{y_i - t_a x_i - t_b}{2\sigma(t_a x_i + t_b)}\right)^2} =$$
$$= \max_{t_a,t_b} \sum_{i=1}^{n} - \left(\frac{y_i - t_a x_i - t_b}{t_a x_i + t_b}\right)^2 =$$
$$= \min_{t_a,t_b} \sum_{i=1}^{n} \frac{1}{(t_a x_i + t_b)^2} \left(t_a x_i + t_b - y_i\right)^2$$

Poslední úprava vypadá opět na použití minimalizace vážené střední kvadratické chyby, koeficienty jsou zde však neznámé. Označme si výslednou sumu L.

Při minimalizaci pomocí parciálních derivací dostaneme

$$0 = \frac{\partial L}{\partial t_a} = \sum_{i=1}^{n} 2 \cdot \frac{t_a x_i + t_b - y_i}{t_a x_i + t_b} \cdot \frac{t_a x_i^2 + t_b x_i - t_a x_i^2 - t_b x_i + x_i y_i}{(t_a x_i + t_b)^2}$$
$$0 = \sum_{i=1}^{n} \frac{(t_a x_i + t_b - y_i)\, x_i y_i}{(t_a x_i + t_b)^3}$$

a

$$0 = \frac{\partial L}{\partial t_b} = \sum_{i=1}^{n} 2 \cdot \frac{t_a x_i + t_b - y_i}{t_a x_i + t_b} \cdot \frac{t_a x_i + t_b - t_a x_i - t_b + y_i}{(t_a x_i + t_b)^2}$$
$$0 = \sum_{i=1}^{n} \frac{(t_a x_i + t_b - y_i)\, y_i}{(t_a x_i + t_b)^3},$$

což však po převedení na společného jmenovatele vede na hledání kořenů polynomů přibližně trojnásobného stupně, než kolika různých hodnot nabývají jednotlivá x_i.

Pokud bychom přesto chtěli pro tento model hledat optimum, máme dvě základní možnosti, jak pokračovat:

- vyřešíme numericky spočtenou soustavu rovnic,
- použijeme iterativní metodu hledání optima.

První možnost vede na použití obecného řešiče, který však bude velmi pomalý, neboť některé soustavy budou na našich datech obsahovat i desetitisíce zlomků.

Můžeme však využít toho, že umíme řešit minimalizaci vážené střední kvadratické chyby: Inicializujeme hledaná t_a, t_b libovolně (např. $t_a = 1, t_b = 1$). Následně spočteme váhy $\frac{1}{(t_a x_i + t_b)^2}$ a vyřešením $\min_{t_a, t_b} L$ s těmito váhami získáme nová t_a, t_b. Tento krok iterujeme do zkonvergování t_a, t_b či do dosažení maximálního povoleného počtu iterací.

Povšimněme si, že má tato metoda pouze lineární zpomalení v počtu iterací oproti výpočtu t_a, t_b dle vah založených na zahlášeném času. Při stabilním počtu iterací je tedy pomalejší pouze o konstantní faktor. Zkonvergování sice nemáme garantováno, nicméně provedené experimenty ukázaly, že k jeho selhání až na výjimky nedochází.

Kombinovaná LR

Přestože by dle výše zmíněných pozorování měla být iterativní LR blíže skutečnosti, bude při ní docházet k nadhodnocování váhy akcí s malým počtem vyrobených kusů a nadprůměrným zahlášeným časem, neboť predikovaný čas je pro takové akce blízký pouze t_b. Experimenty v části 7.2 ukazují, že tento problém způsobuje, že výsledky iterované LR jsou srovnatelné s váženou LR.

Můžeme však využít skutečnosti, že vážená LR a iterativní LR mají problémy s odlišnými typy akcí. Zvolme parametr $\alpha \in [0, 1]$. Pokud použijeme iterativní metodu, ovšem jako váhy budeme

$$\alpha \cdot \frac{1}{y_i^2} + (1 - \alpha) \cdot \frac{1}{(t_a x_i + t_b)^2},$$

měly by být výsledky pro vhodné α lepší, což se následně v 7.2 prokázalo.

Optimální LR

Všechny dosud popsané LR mají nevýhodu kvůli aproximaci chybové funkce err. Především pro nalezení nejlepší možné predikce T pro testovací množiny (a tedy dolního odhadu hodnoty ztrátové funkce) potřebujeme regresor schopný nalézt minimum vzhledem k zadané chybové funkci.

Obecně tento problém nelze vyřešit bez kontroly všech možných kombinací $t_a(V), t_b(V)$ pro všechny operace $V \in \mathcal{V}$. Pokud ale budeme předpokládat, že se zadaná ztrátová funkce chová „rozumně“ (tj. například má jediné lokální minimum které je současně globální, je diferencovatelná, její gradient v omezeném intervalu pro $t_a(V), t_b(V)$ neklesá kromě minima k 0, je konvexní či alespoň gradient směřuje ve většině případů od minima), můžeme použít numerickou optimalizaci pro nalezení optimálních $t_a(V), t_b(V)$ pro všechny operace.

Funkce

$$loss = \sum_{a \in \mathcal{A}} \left(\ln\Big(t_a\big(V(a)\big) n(a) + t_b\big(V(a)\big) \Big) - \ln\big(t\,(a) \big) \right)^2$$

s proměnnými t_a a t_b by však naštěstí měla většinu těchto vlastností splňovat. Můžeme si navíc povšimnout, že lze optimalizovat hodnoty $t_a(V), t_b(V)$ pro každou operaci $V \in \mathcal{V}$ zvlášť, neboť členy součtu pro různé operace jsou nezávislé. Experimenty ukázaly, že i naivní způsob iterativní změny t_a, t_b rychle konverguje k minimu – detaily probereme v kapitole 7.

4.2.3 Theil-Senův regresor

Všechny předchozí regresory mají nevýhodu v tom, že jediný chybný zahlášený čas může výslednou predikci libovolně ovlivnit. Jelikož však vstupní data obsahují řadu chyb, nemáme garantováno, že případná detekce outlierů bude mít perfektní účinnost. Potřebovali bychom tedy model schopný určitou míru chybovosti akceptovat. Této schopnosti se říká (statistická) robustnost, definovaná v článku Rousseeuw a Leroy (1987).

Definice 7 (Bod selhání, Robustnost)**.** *Buď $n, m \in \mathbb{N}$, $m \leq n$ a*

$$X = \Big\{ (\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n) \Big\} = \Big\{ \big((x_{11}, \ldots, x_{1k}), y_1 \big), \ldots, \big((x_{n1}, \ldots, x_{nk}), y_n \big) \Big\}$$

množina dat, pro které je prováděna regrese R, regresor natrénovaný na množině X pak budeme označovat R_X a jeho predikci pro bod x budeme označovat $R_X(x)$. Buď dále X' množina

$$X' = \Big\{ \big(\boldsymbol{x'_1}, y'_1 \big), \ldots, \big(\boldsymbol{x'_m}, y'_m \big) \Big\}$$

tvořená libovolně zvolenými m prvky stejného typu jako prvky X. Za X_m poté označme množinu vzniklou z X, ve které m prvků nahradíme prvky z množiny X'. Pro $\delta \in [0, 1]$ označme

$$bias(R, X, \delta) = \sup_{X' : |X'| \leq \delta \cdot |X|, (\boldsymbol{x'}, y') \in X_m} \Big| R_X(\boldsymbol{x'}) - R_{X_m}(\boldsymbol{x'}) \Big| .$$

Jako bod selhání *regresoru R pro $n \in \mathbb{N}$ pak budeme označovat*

$$\min_{\delta>0} \left(\exists X, |X| = n : bias(R, X, \delta) = \infty\right).$$

Za robustnost *regresoru R označíme hodnotu bodu selhání R pro $n \to \infty$.*

Definice nám říká, jak velké množství dat musíme poškodit, abychom si vynutili libovolnou změnu výsledku regrese. V našich datech lze očekávat $1 - 5\%$ výrazněji poškozených záznamů akcí (buďto s chybným výrobním časem nebo s nesprávným popisem), model s touto či vyšší robustností pro velké n by tedy měl dávat lepší výsledky. Změna jediného bodu dokáže predikci libovolné z doposud popsaných LR neomezeně změnit, jejich bod selhání je tak roven $1/n$ a robustnost 0.

Poznámka. Povšimněme si, že robustnost nikdy nemůže být vyšší než 0,5 (kromě regresoru predikujícím konstantu bez ohledu na vstupní data – ten má *bias* vždy 0, bod selhání tak pro něj je z definice vždy $+\infty$). Při změně alespoň poloviny vstupních dat totiž již nemůžeme rozlišit, zda namísto nich nebyla pozměněna všechna zbylá data. Kupříkladu libovolná regrese pro data bez nezávislých proměnných a dvěma body v X nemůže predikovat žádnou hodnotu bez selhání, neboť kterýkoli z těchto dvou bodů mohl být změněn a vzdálenost predikce alespoň od jednoho z nich je neomezeně vysoká.

Robustních lineárních regresorů s různými hodnotami robustnosti je známo mnoho, viz práce Bai (2012). My se zaměříme především na Theil-Senův regresor.

Hlavní myšlenkou Theil-Senova regresoru je využití mediánu namísto průměrů – libovolná změna malé části dat nemůže medián neomezeně změnit. Zaměříme se na variantu s jednorozměrnou nezávislou proměnnou, v našem případě počet kusů akcí dané operace

Definice 8 (Theil-Senův regresor pro jednu nezávislou proměnnou)**.** *Mějme dána vstupní data $((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n))$. Poté predikované t_a bude rovno*

$$t_a = median\left\{\forall i, j \in [n], i \neq j, x_i \neq x_j : \frac{y_i - y_j}{x_i - x_j}\right\}.$$

Dvojice se stejnou x-ovou souřadnicí vynechávejme, neboť vertikální přímky mají směrnici $\pm\infty$, tudíž je můžeme považovat z poloviny větší i menší než libovolnou zvolenou hodnotu a nemají tedy vliv na určení mediánu.

Poté, co je určeno t_a, dopočteme t_b tak, aby výsledná přímka dělila data na poloviny, tj.

$$t_b = median\left\{\forall i : y_i - t_a x_i\right\}.$$

Povšimněme si, že nelze použít průměr ani v tomto kroku, neboť změnou jednoho y_i by bylo možné libovolně upravit t_b.

Tento regresor je vhodný z řady důvodů:

- Jedná se o bezparametrickou metodu,

- Pro jednu nezávislou proměnnou dosahuje dobré robustnosti $1 - \frac{1}{\sqrt{2}} \approx 0{,}29$[1] – vzhledem ke kvalitě vstupních dat je tato hodnota zcela dostačující, na rozdíl od jiných regresorů garantujících vyšší hodnoty robustnosti navíc neignoruje tak velkou část dat,
- Lze (dle Cole a kol., 1989) spočítat se složitostí $O\left(n\log\left(n\right)\right)$. Pro velké množiny lze také dobře aproximovat vzorkováním.

Nevýhodou je nemožnost úpravy dle použité ztrátové funkce, regresor kvůli používání mediánu předpokládá shodné počty zahlášených časů pod a nad skutečným výrobním časem. Rozložení *err* toto sice splňuje, nicméně pro operace s málo akcemi je vysoká pravděpodobnost jejich odchýlení.

[1] Existuje $\binom{n}{2} \approx \frac{n^2}{2}$ dvojic bodů, tedy i potenciálních směrnic. Pro ovlivnění mediánu jich tak musíme změnit alespoň $\frac{n^2}{4}$, přičemž změna jednoho bodu může změnit jen směrnice přímek jím vedoucích. Při změně $\left(1 - \frac{1}{\sqrt{2}}\right) n$ bodů tak nezměníme více než

$$\left(1 - \frac{1}{\sqrt{2}}\right) \cdot n^2 - \left(1 - \frac{1}{\sqrt{2}}\right)^2 \cdot \frac{n^2}{2}$$

směrnic (musíme odečíst dvakrát započítané s oběma změněnými body), což je po zjednodušení $\frac{n^2}{4}$, neboli přesně naše hraniční hodnota.

5. Clusterizace dat metodou učení bez učitele – Latent Dirichlet allocation

5.1 Motivace

Hlavní nevýhodou lineární regrese je velikost množiny $\mathcal{V}$ – pro mnoho operací máme k dispozici pouze jednu či několik akcí, což není dostatečný počet pro její použití. Pokud by se nám tedy podařilo rozdělit operace do několika početných skupin $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k$, kde operace ve skupinách budou mít podobné vlastnosti (k je hyperparametr), následná regrese nad těmito množinami bude mnohem méně citlivá na šum v datech. Budeme navíc schopni provést predikci i pro zcela nové operace a vzhledem k velkému počtu akcí v každé skupině i odhadnout očekávanou chybu predikce.

Tomuto seskupování akcí budeme říkat *clusterizace*. Vzhledem k tomu, že ke clusterizaci využijeme pouze atributy operací a závisející produkty (tj. nevyužijeme jejich zahlášené akce), patří použitá metoda do skupiny algoritmů označovaných jako učení bez učitele (unsupervised learning).[1]

Vzhledem k tomu, že operace mají několik textových popisů, potřebujeme využít clusterizační algoritmus, který s nimi umí pracovat. Očekáváme například, že operace, které využívají ke svému popisu stejná slova a jsou prováděna na stejných pracovištích, budou mít podobné výrobní časy. Jako použitou metodu zvolíme *Latent Dirichlet alocation* (Blei a kol., 2003), zkráceně LDA. Ta se využívá zejména pro clusterizaci textových dokumentů dle jejich obsahu a v této oblasti dosahuje velmi dobrých výsledků.

5.2 Algoritmus LDA

Algoritmus LDA je založen na hledání nejpravděpodobnějšího rozdělení jednotlivých slov dokumentů do clusterů vzhledem k předpokladu jejich Dirichletova rozdělení s předem určenými hyperparametry.

Definice 9 (Dirichletovo rozdělení)**.** *Nechť* $\boldsymbol{x} = x_1, x_2, \ldots, x_k \in (0,1)^k$ *splňují* $\sum_{i=1}^{k} x_i = 1$*. Pak pro pevné* $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k) \in \left(R^+\right)^k$ *bude*

$$Dir(\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{k} x_i^{\alpha_i - 1}$$

označovat Dirichletovo rozdělení *s parametrem* $\boldsymbol{\alpha}$*, kde normalizační konstanta*

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{k} \alpha_i\right)}$$

[1] Jelikož následně na vytvořených clusterech používáme lineární regresi, tedy supervised metodu, není označení jako unsupervised learning zcela přesné. Samotná clusterizace, což je klíčová část algoritmu, však toto označení splňuje.

odpovídá vícerozměrné beta funkci.

V případě, že si budou všechna α_i rovna, budeme pro $\alpha > 0$ používat pouze $Dir(\alpha)$ namísto $Dir\left((\alpha, \alpha, \ldots, \alpha)\right)$.

Využijeme-li aritmeticko-geometrickou nerovnost pro prvky $\boldsymbol{x}$, dostaneme

$$\prod_{i=1}^{k} x_i \leq \left(\frac{\sum_{i=1}^{k} x_i}{k} \right)^k = k^{-k}$$

s rovností nastávající pouze pro $x_i = \frac{1}{k}$. Povšimněme si, že $\prod_{i=1}^{k} x_i^{\alpha-1}$ je tak podle hodnoty α vždy jednoznačně porovnatelné s $k^{-k(\alpha-1)}$ umocněním celé nerovnosti na $\alpha - 1$.

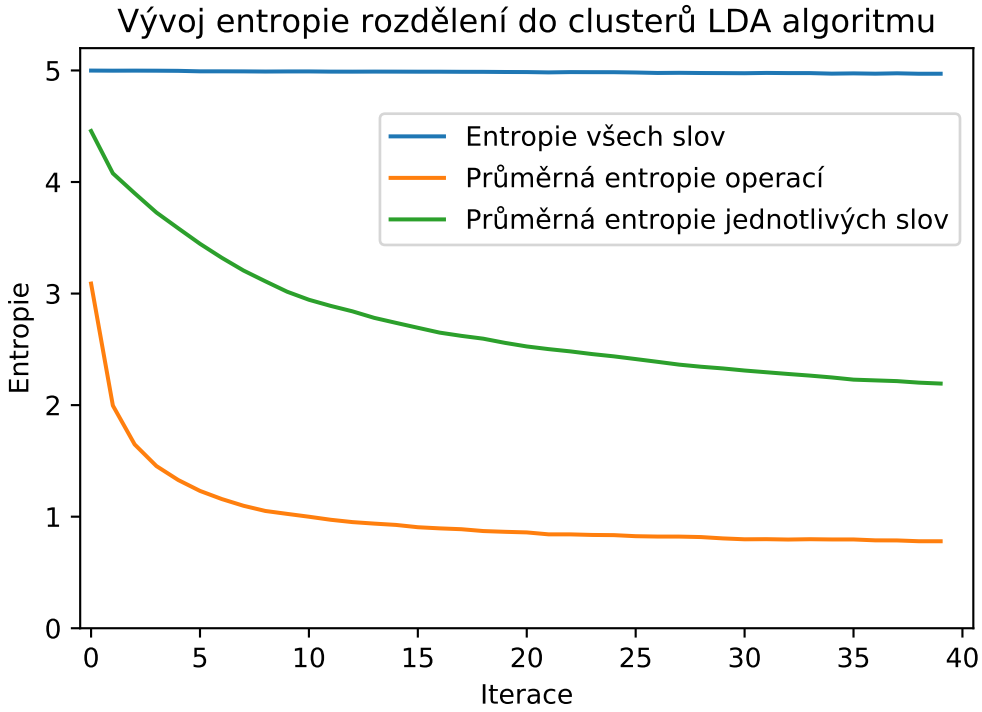
Důsledkem je, že $Dir(\alpha)$ nabývá pro $\alpha > 1$ maxima pro všechna $x_i = \frac{1}{k}$ a klesá s nerovnoměrností prvků $\mathbf{x}$. Pro $\alpha < 1$ naopak pro všechna $x_i = \frac{1}{k}$ nabývá minima a pro $\alpha = 1$ jde o rovnoměrné rozdělení, neboť $x_i^0 = 1$ pro libovolné $x_i \in (0, 1)$.

Základními myšlenkami modelu LDA je použití Dirichletova rozdělení s různými parametry pro distribuci jednotlivých dokumentů mezi clustery:

- Jednotlivé clustery nejsou předem rozlišitelné (tj. pravděpodobnost určitého rozdělení je stejná nezávisle na permutaci čísel clusterů),
- Clustery obsahují podobné množství slov (tj. pravděpodobnost relativních zastoupení x_i jednotlivých clusterů budeme modelovat jako $Dir(\alpha)$ s nějakým $\alpha > 1$),
- Většina výskytů konkrétního slova by měla být přiřazena do jednoho clusteru (tj. $x_{i,w}$ odpovídající relativnímu zastoupení clusteru i pro slovo w modelujeme s nějakým $\alpha < 1$),
- Většina slov jednoho dokumentu by měla být ze stejného clusteru (tj. $x_{i,d}$ odpovídající relativnímu zastoupení clusteru i slovy dokumentu d modelujeme s $\alpha < 1$ ne nutně shodným s předchozím bodem).

Optimální řešení lze nalézt vyzkoušením všech možných rozdělení $\mathcal{V}$ na clustery a ohodnocením jejich věrohodnosti dle tohoto modelu. Vzhledem k jejich exponenciálnímu počtu však není takové řešení možné – použijeme tedy iterativní zlepšování pomocí Gibbsova vzorkování. Průběh vývoje jednotlivých entropií na reálných datech můžeme vidět na obrázku 5.1.

Ukázalo se, že kvalita rozdělování operací do clusterů dle podobnosti jejich popisu je poměrně vysoká. Výsledky pro predikci získané při krosvalidaci však ukázaly, že tento přístup není pro reálná data dostatečně dobrý (detaily viz kapitola 7.3). Nebudeme se tedy dále zabývat hledáním potenciálních vylepšení tohoto modelu pro řešení našeho problému, neboť bez opuštění unsupervised learningu zde nelze dosáhnout výsledků srovnatelných s ostatními modely.



Obrázek 5.1: Vývoj entropie slov operací pro $k = 30$ při hledání optima LDA pomocí Gibbsova vzorkování

6. Predikce pomocí neuronových sítí

S růstem výpočetního výkonu se jednou z hlavních metod pro predikci staly umělé neuronové sítě. Zejména pak tzv. hluboké učení, které využívá sítě s více vrstvami, přičemž struktura jednotlivých vrstev se liší v závislosti na jejich účelu. Jako dobrý zdroj informací o konstrukci sítí, výpočtech v nich a jejich vlastnostech lze využít obsáhlou knihu „Deep Learning“(Goodfellow a kol., 2016).

Tato metoda má oproti předchozím řadu výhod, zejména pak:

- Schopnost aproximace libovolné spojité funkce,
- výborné výsledky v klasifikačních i regresních úlohách, pokud není známý vztah mezi nezávislými a závislými proměnnými,
- schopnost extrakce relevantních informací z textových řetězců. Na rozdíl od algoritmu LDA není síť vázaná na jednotlivá slova, ale může hledat i kombinace slov či jen část slova, pokud se ukáže, že jsou pro predikci klíčové – zejména u flektivních jazyků jako je čeština může být nárůst užitečnosti výrazný vzhledem k počtu variant stejného slova,
- možnost dávat odhad spolehlivosti predikovaného času.

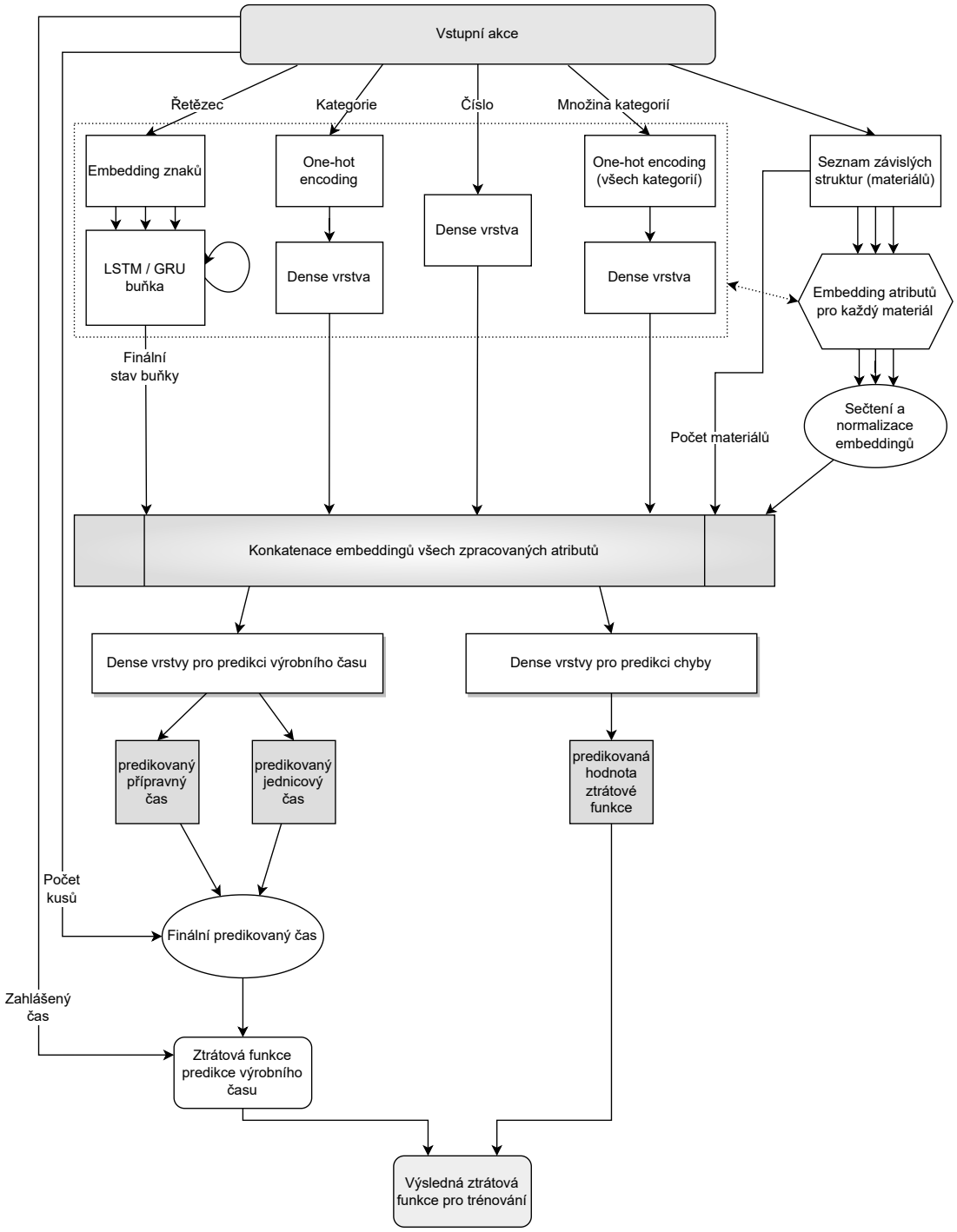
Bohužel, oproti jednodušším metodám jako je LR však přinášejí i řadu problémů. Mezi nejdůležitější patří:

- Vysoká výpočetní náročnost,
- jedná se o parametrický model – je nutné ruční nastavení architektury sítě a jejích hyperparametrů[1],
- snadné přeučování na trénovací množinu kvůli množství stupňů volnosti,
- obtížné zdůvodnění volby jednotlivých predikcí.

6.1 Zpracování vstupu

Nejprve musíme vyřešit, jakým způsobem zakódujeme vstupní data (tzv. embedding) tak, aby byla použitelná jako vstup vytvářené sítě. Rozebereme postupy zpracování pro jednotlivé typy informací. Výsledné schéma sítě použité pro experimenty můžeme nalézt na obrázku 6.1.

[1] Existují i algoritmy, které jsou schopny architekturu vytvořit, například NEAT (Stanley a Miikkulainen, 2002). Vzhledem k množství dat a komplexním datovým typům (řetězce, materiály) však není jejich použití příliš vhodné.



Obrázek 6.1: Schéma výsledné neuronové sítě

6.1.1 Reálná čísla

Mějme číslo $r \in \mathbb{R}$. Tento druh atributu je přímočarý, neboť neuronová síť pracuje s reálnými čísly ve svých neuronech. Pokud tedy nenabývá atribut r jen malého množství různých hodnot (v takovém případě může být vhodnější využít jeho zakódování do kategorií), můžeme r použít přímo.

Vytvoříme jeden vstupní neuron, jemuž nastavíme hodnotu r. Ten spojíme s vrstvou dalších neuronů, čímž umožníme nelinearitu výstupu vzhledem k hodnotě r. Volitelně můžeme přidat další vrstvy neuronů plně propojené s předchozí vrstvou (tzv. *dense* vrstvy) pro zvýšení obecnosti embeddingu, ovšem můžeme tím zpomalit učení. Poslední vrstvu budeme následně považovat za cílový embedding čísla r.

Vzhledem k tomu, že učení neuronové sítě je založeno na výpočtu gradientu, je vhodné číselné hodnoty normalizovat. Pokud určitý atribut nabývá u operací hodnot v řádu tisíců, bude gradient na jeho změny mnohem citlivější než na atributy s hodnotami v řádu jednotek. Stabilizace vah pro nenormalizovaný atribut by tak trvala výrazně déle a lze předpokládat, že by negativně ovlivnila rychlost učení.[2]

6.1.2 Kategorická data

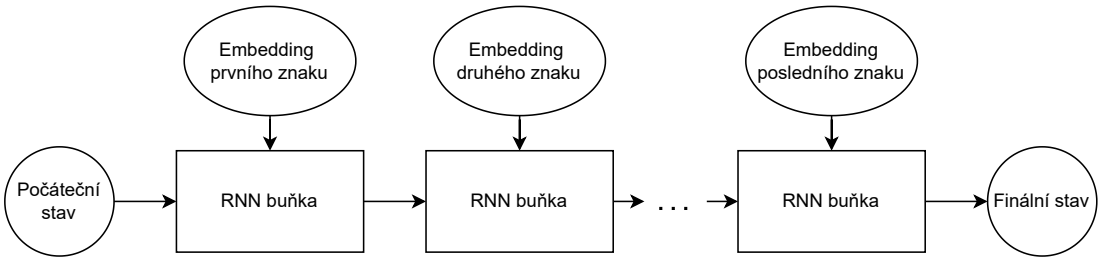
Pokud může daný atribut nabývat pouze hodnot z malé množiny C a není zřejmá souvislost mezi jednotlivými hodnotami, budeme prvky C nazývat kategoriemi. Pro každou kategorii vytvoříme jeden neuron. Má-li vstupní informace hodnotu $c \in C$, nastavíme hodnotu c-tého neuronu na 1 a zbylým na 0, tedy tzv. one-hot kódování. Obdobně jako v předchozím případě tyto neurony spojíme s jednou či více dense vrstvami pro zisk finálního embeddingu.

Pro tento typ atributů nepotřebujeme provádět dodatečnou normalizaci, neboť je zaručena z definice. Často se však stává, že mnoho kategorií je zastoupeno pouze v několika záznamech. Nemáme tedy dostatek dat pro jejich relevantní využití, navíc výrazně zvyšují počet vstupních neuronů a tím i snižují rychlost učení a tendenci se přeučovat. Tento problém vyřešíme tak, že všechny kategorie s méně než n_{min} záznamy sloučíme do jediné kategorie „unknown“, přičemž n_{min} je hyperparametr. Užitečnost takové kategorie pro predikci sice nemůžeme zaručit, ovšem počet vstupních neuronů takto znatelně omezíme a snížíme pravděpodobnost přeučování pro atributy sloučené do kategorie „unknown“.

6.1.3 Množiny kategorií

Atributy tohoto typu zobecňují kategorická data – každý záznam nyní obsahuje libovolnou podmnožinu kategorií $S \subseteq C$. Použijeme stejný postup jako v sekci 6.1.2, ovšem nyní nastavíme hodnotu 1 pro všechny neurony, které odpovídají kategoriím $s \in S$.

[2] Efekt na zbytek sítě nelze jednoznačně určit, záleží na použitém učícím algoritmu a na dalších vrstvách sítě – pokud například výstupy tohoto atributu vedou pouze přes neurony se sigmoidou jako aktivační funkcí, do zbytku sítě budou hodnoty omezeny vždy na interval $(0, 1)$. Takový atribut by pak neměl mít nechtěný vliv, v nejhorším případě budou tyto výstupy ignorovány jako irelevantní.



Obrázek 6.2: Schéma zpracování řetězce pomocí RNN buňky

Normalizaci je v tomto případě možné provést, nicméně obvykle se neprovádí, jelikož kategorií bývá nejvýše několik desítek. Atributy tohoto typu také obvykle obsahují pouze malé podmnožiny C, vynechání normalizace tedy nezpůsobí velké problémy s gradientem.

6.1.4 Řetězce

Řetězce jsou tvořeny posloupností znaků, kde záleží na hodnotách jednotlivých znaků i na jejich pořadí. Znaky odpovídají kategorickým datům, můžeme na ně tedy použít algoritmus ze sekce 6.1.2. Povšimněme si, že je třeba použít shodné zakódování pro všechny znaky, tzv. character-level embedding. Pro zlepšení učení můžeme také provést dodatečnou analýzu vstupních řetězců a hledat častá slova či sekvence znaků – takové sekvence pak můžeme nahradit nově vytvořeným znakem, tzv. word-level embedding. Tím zkrátíme délku řetězců a zjednodušíme tak síti udržení nalezené informace během jejich zpracovávání za cenu zvětšení množiny znaků.

Řetězce mají oproti předchozím typům atributů dvě speciální vlastnosti – záleží na pořadí znaků a jejich délka je proměnlivá. Nemůžeme tedy snadno vytvořit blok statické velikosti, který by byl schopen zpracovat libovolný řetězec[3]. K řešení využijeme techniky rekurentních neuronových sítí (RNN).

Tyto sítě jsou navrhnuty pro zpracování sekvencí proměnlivé délky a jsou založené na pevné výpočetní buňce. Buňka je nejprve nakopírována tolikrát, jaká je délka řetězce a následně je její vstup složen z embeddingu odpovídajícího znaku řetězce a výstupu předchozí buňky, viz obrázek 6.2. Za finální embedding řetězce pak považujeme výstup poslední buňky.

Toto schéma nijak neinformuje buňku o tom, že aktuálně zpracovávaný znak je poslední a mohlo by být užitečné provést atypické změny ve finálním stavu. Často se tak používá přidání speciálního nového typu znaku end-of-word (EOW) na konec řetězce. Obdobně ne každý typ RNN buňky používá trénování počátečního stavu, může proto být vhodné přidat na začátek řetězce nový „inicializační“ znak beginning-of-word (BOW). Přidat nový znak na oba konce řetězce je vhodné také proto, že se obvykle řetězec pomocí RNN zpracuje i ve zpětném pořadí – RNN buňky mají obecně problém s tendencí zapomínat důležité informace o řetězci, jsou-li založeny na takové části řetězce, po které následuje ještě mnoho dalších

[3]Pokud bychom pevně omezili délku každého řetězce, bylo by to teoreticky možné. Ovšem embedding znaku na každé pozici by pak musel být s výstupní vrstvou propojen s vlastními vahami, aby zůstala zachována informace o pozici. Ani taková úprava by však nebyla příliš účinná, neboť posun většiny řetězce o jediný znak (například změnou nějakého čísla na začátku z 9 na 10) by způsobilo zcela jiný výsledný embedding.

znaků. Speciálně tedy dopředný průchod hůře využívá počátek a zpětný průchod konec vstupního řetězce.

Pro experimenty budeme využívat RNN buňky dvou standardních typů. Prvním typem bude *Gated recurrent unit* (Cho a kol., 2014, GRU), která s každým znakem aktualizuje část vnitřního stavu (založenou na hodnotách vnitřních neuronů) dle nového znaku . Druhým typem pak *Long short-term memory* (Hochreiter a Schmidhuber, 1997, LSTM), která je založena na podobné myšlence jako GRU, nicméně má navíc interní stav buňky, který není součástí výstupu. Může tak lépe uchovávat dlouhodobější informace, neboť při učení lépe brání postupnému klesání gradientu při zpětné propagaci.

6.1.5 Neuspořádané seznamy závislých struktur

Operace či produkty mohou být závislé na variabilním počtu dalších strukturovaných proměnných. V našem případě může mít každý produkt přiřazen libovolný nezáporný počet dalších produktů, které odpovídají jeho materiálům. Pro každý produkt tak můžeme navíc mít seznam informací libovolného z výše popsaných typů, například rozměry využitých materiálů.

Jednotlivé prvky seznamu můžeme zpracovávat standardně dle jejich typu, ovšem je pak třeba složit jejich embeddingy dohromady, neboť do zbylé části sítě potřebujeme mít každý atribut zakódovaný do pevné délky. Tento problém lze vyřešit různými způsoby, zmiňme zde několik návrhů:

- Sečtení jednotlivých embeddingů. Toto řešení je celkem přímočaré, musíme si však dát pozor na nárust hodnot s počtem materiálů. Je tedy vhodné provést určitý typ normalizace, například volba pevného koeficientu α založeného na očekávaných počtech materiálů, kterým součet embeddingů nakonec přenásobíme.

- Použití průměru jednotlivých embeddingů pro každý atribut. Tento přístup řeší problém nárustu hodnot z prvního návrhu, má však problém s nestabilním výstupem na základě počtu materiálů. Obsahuje-li nějaký materiál klíčovou informaci interpretovanou konkrétní hodnotou některých prvků embeddingu, průměrování může tyto hodnoty libovolně přeškálovat dle počtu materiálů i kdyby zbylé materiály měly pro tyto prvky nulové hodnoty. Lze tedy očekávat výrazně horší uchovávání informací.

- Použití maxima pro každý prvek embeddingu. Toto řešení nemá problémy předchozích dvou přístupů, neboť hodnoty výsledného embeddingu jsou omezeny jednotlivými embeddingy a pokud v embeddingu nějakého materiálů bude určitý prvek nastaven na vysokou hodnotu, zůstane tato hodnota zachována nehledě na počet materiálů. Nevýhodou je horší učení (nenulový gradient vyjde pouze pro maximální prvky) a situace, kdy zakódování klíčové informace využívá hodnoty více prvků embeddingu. V takovém případě složení maxima z různých materiálů může vést k vzájemnému přepsání částí klíčových informací.

Použití vstupů tohoto typu může být problematické vzhledem k vyšší citlivosti na nastavení parametrů a výraznému nárůstu výpočetního času kvůli dodatečnému rozměru, zejména budeme-li využívat i složitější typy dat – má-li například

nějaký produkt 10 materiálů, použití atributu řetězcového typu bude vyžadovat provedení RNN embeddingu pro každý řetězec, zpomalující zpracování až desetinásobně v závislosti na ostatních využitých atributech.

6.2 Predikce časů pro danou akci

Mějme na vstupu akci a. V neuronové síti nejprve vytvoříme embeddingy všech požadovaných atributů z a dle pravidel z předchozí sekce 6.1. Následně všechny tyto embeddingy spojíme za sebe do jedné vrstvy. Tuto vrstvu budeme nazývat jako základní.

Základní vrstvu následně propojíme s jednou či více dense vrstvami pro provázání informací z jednotlivých embeddingů. nakonec vytvoříme dva neurony propojené se všemi výstupy poslední dense vrstvy, které budou predikovat $t_b\left(V\left(a\right)\right)$ a $t_a\left(V\left(a\right)\right)$. Finální predikci sítě pak budeme označovat jako $N(a)$.

Jelikož skutečná t_b a t_a mohou nabývat libovolných nezáporných hodnot, je pro tyto dva neurony třeba zvolit vhodnou aktivační funkci. Vzhledem k rozsahu hodnot výrobních časů není volba správné funkce zřejmá a může být závislá na vstupních datech. Během experimentů proto můžeme vyzkoušet několik různých variant.

- Lineární, zdola omezená nulou ($f(x) = \max\left(0, x\right)$). U této varianty nejspíše bude problém s dobrou predikcí v případech, kdy skutečný přípravný či jednicový čas je blízko nule či naopak velmi vysoký. Pro standardní situace se však může učit rychleji vzhledem ke konstantní derivaci pro libovolné kladné x.

- Exponenciální ($f(x) = \mathrm{e}^x$). Tato varianta by měla být schopna dobře predikovat pro velmi malé i velmi velké skutečné výrobní časy t_b, t_a. Problémem může být riziko občasného výskytu velké chyby v predikci pro vyšší hodnoty t_b či t_a – pokud se například na vstupu tohoto neuronu místo správného $x = 5$ vyskytne $x = 10$, hodnota err^2 pro takovou predikci bude blízko 25 bez ohledu na predikci pro zbývající člen z t_b, t_a. Takové situace mohou mít kvůli skokové změně gradientu negativní vliv na rychlost učení.

- Softplus ($f(x) = \ln\left(1 + \mathrm{e}^x\right)$). Tuto funkci lze považovat za kombinaci předchozích dvou. Pro $x > 2$ člen e^x výrazně dominuje člen 1, jeho zanedbáním tedy dostaneme $f(x) \approx \ln\left(\mathrm{e}^x\right) = x$. Pro $x < -2$ je hodnota e^x blízká nule a můžeme tak využít aproximace $\ln\left(1 + y\right) \approx y$ platné pro y blízké nule. Dosazením za y dostaneme $f(x) \approx \mathrm{e}^x$. Na intervalu $\langle -2, 2\rangle$ hodnota funkce softplus hladce přechází mezi těmito dvěma funkcemi.

  Tato funkce je tedy vhodná pro dobrou predikci t_b či t_a blízkých nule, zároveň pro ni nenastává problém s nárůstem chyby pro vysoké výrobní časy. Může však mít problém s jejich dobrou predikcí vzhledem ke sdílení této vlastnosti s lineární funkcí.

- Druhá mocnina softplus ($f(x) = \ln^2\left(1 + \mathrm{e}^x\right)$). Jedná se o pokus vyřešit zbývající problém funkce softplus, kdy zjednodušíme predikci pro vysoká t_b, t_a za cenu snížení přesnosti predikce pro záporná x přibližně na polovinu. Pro kladná x bude nyní $f(x) \approx x^2$, tedy kvadraticky rostoucí rozsah

predikovatelných hodnot. Jelikož platí $e^x = \left(e^{x/2}\right)^2$, pro záporná x bude druhá mocnina softplus přibližně dvakrát citlivější na hodnotu argumentu než klasická funkce softplus.

6.3 Metoda učení

Chceme nalézt takové parametry sítě, pro které budou $N(a)$ nejlépe odpovídat skutečným výrobním časům T. Potřebujeme tedy minimalizovat odpovídající ztrátovou funkci *loss* určenou v kapitole 3.2

$$\sum_{a \in \mathcal{A}} \left(\ln\left(N\left(a\right)\right) - \ln\left(t\left(a\right)\right) \right)^2 .$$

Jelikož je zahlášený čas $t\left(a\right)$ pro každou akci pevný a $N\left(a\right)$ je jen lineární funkcí predikovaných $t_b\left(V\left(a\right)\right), t_a\left(V\left(a\right)\right)$, je *loss* diferencovatelná, lze tedy určit $\frac{\partial loss}{\partial N(a)}$ pro každou akci a a můžeme využít různých gradientních metod pro hledání optimálních vah v síti.

Učení bude probíhat tak, že si v každé iteraci nejprve rozdělíme trénovací data na bloky. Vzhledem k množství akcí je problematické spočítat gradient pro celou trénovací množinu současně, budeme tedy postupně počítat a aplikovat gradienty pro jednotlivé bloky. Vzhledem k rozdílům mezi bloky tento postup rovněž snižuje pravděpodobnost zastavení se v lokálním minimu. Pro dense vrstvy také můžeme použít dropout (tj. vynechání náhodné podmnožiny jejich neuronů zvolené velikosti) kvůli regularizaci.

Na samotné spočtení gradientu lze využít algoritmus zpětné propagace, popsaný v článku „Learning representations by back-propagating error“ (Rumelhart a kol., 1986). Můžeme následně přímo provést změnu vah v síti pomocí standardní gradient-descent metody (tj. iterované změny všech vah sítě o předem zvolený násobek η gradientu, neboli *learning rate*), nebo využít pokročilejších algoritmů, které pro běžné problémy řešené pomocí neuronových sítí obvykle dávají lepší výsledky. Myšlenkou těchto algoritmů je využití dříve spočtených gradientů pro úpravu aktuálního kroku. Kupříkladu momentová metoda (Qian, 1999) využívá „hybnost“ změn vah z poslední iterace a pro následující iteraci použije konvexní kombinaci předchozí iterace a aktuálního gradientu. Alternativním přístupem jsou adaptivní algoritmy schopné automaticky snižovat learning rate jednotlivých vah v závislosti na tom, jak výrazně již byla tato váha modifikována. Tyto varianty vycházejí především z algoritmu Adagrad (Duchi a kol., 2011).

Popsané přístupy lze různými způsoby modifikovat a kombinovat pro uchování výhod jednotlivých algoritmů. V našich experimentech využijeme algoritmus „Adaptive Moment Estimation“ (Kingma a Ba, 2015), zkráceně Adam. Ten je založený na myšlence algoritmu Adagrad, ovšem namísto postupného snižování learning rate jsou koeficienty pro jednotlivé váhy založeny na prvním a druhém momentu předchozích gradientů spočtených pro tyto váhy. Jsou-li tedy gradienty pro danou váhu stabilní, velikost změny bude blízká η. Jsou-li však nestabilní, druhý moment způsobí sníží learning rate pro tuto váhu, čímž napomůže stabilizaci. Tato metoda v praxi dosahuje dobrých výsledků, proto jsme ji použili i pro naše řešení.

6.4 Rozšíření sítě o predikci očekávané chyby predikovaných časů

Rozptyly zahlášených časů se pro jednotlivé operace výrazně liší. Při použití vytvořené sítě v reálné situaci by tedy bylo užitečné získávat i očekávanou kvalitu predikce pro určení vhodnosti použití predikovaného času pro plánování, případně pro volbu vhodných délek rezervních intervalů mezi jednotlivými operacemi.

Do vytvořené sítě chceme tedy nějakým způsobem přidat neuron predikující očekávanou chybu – základní vrstva by měla obsahovat zpracované všechny vstupní informace, dodatečnou podsíť tedy spojíme s touto vrstvou. Obdobně jako pro predikci přidáme jednu či více dense vrstev pro zpracování jednotlivých informací a výstup poslední z nich bude použit pro neuron predikující přesnost, viz schéma 6.1. Výstup tohoto neuronu pro akci a označme jako $L(a)$.

Potřebujeme najít vhodný způsob, jakým provážeme neurony predikující přípravný a jednicový čas a nový neuron predikující přesnost. Pro řešení tohoto problému využijeme techniku z článku „Pairing an arbitrary regressor with an artificial neural network estimating aleatoric uncertainty“ (Gurevich a Stuke, 2019). Hlavní myšlenkou je modifikace použité ztrátové funkce tak, aby její gradient vždy odpovídal modifikaci vah v neuronu predikující nepřesnost vůči hodnotě původní ztrátové funkce.

Pro neuron predikující přesnost použijme sigmoidu jako aktivační funkci, aby $L(a)$ byla garantována v intervalu $(0, 1)$. Jako novou ztrátovou funkci $loss'$ pak použijme

$$loss' := \sum_{a \in \mathcal{A}} \left(-loss\left(N(a), t(a)\right) \cdot \ln\left(1 - L(a)\right) - \lambda \ln L(a) \right),$$

kde λ je hyperparametr.

Lze ukázat, že pro zafixovanou hodnotu $loss\left(N(a), t(a)\right)$ má tato funkce pro $L(a) \in (0, 1)$ jediné lokální a současně globální minimum (viz článek). Pokud tedy provedeme parciální derivaci této funkce dle $L(a)$ pro nějakou akci a a položíme ji rovnou 0 pro nalezení minima, dostaneme

$$0 = \frac{\partial loss'}{\partial L(a)} = \frac{loss\left(N(a), t(a)\right)}{\left(1 - L(a)\right)} - \frac{\lambda}{L(a)},$$

$$loss\left(N(a), t(a)\right) = \frac{\lambda\left(1 - L(a)\right)}{L(a)}.$$

Při učení sítě tedy bude $L(a)$ konvergovat k takové hodnotě, aby $\frac{\lambda\left(1-L(a)\right)}{L(a)}$ konvergovala k aktuální hodnotě $loss\left(N(a), t(a)\right)$. Hodnota $loss\left(N(a), t(a)\right)$ se sice v průběhu učení mění, čímž se bude měnit i $L(a)$, nicméně po stabilizaci sítě by pro každou akci měly výsledky téměř přesně splňovat tuto rovnost.

Zbývá zjistit, jaký vliv má tato úprava na původní gradienty, neboli porovnání

$$\frac{\partial loss'}{\partial N(a)} = -\frac{\partial loss}{\partial N(a)} \cdot \ln\left(1 - L(a)\right),$$

změnou je tedy přenásobení původního gradientu koeficientem $-\ln\Big(1-L(a)\Big)$. Tento koeficient znamená, že akce operací s permanentně špatnou predikcí budou mít nižší vliv na učení, což by mělo zlepšit predikci pro operace se zahlášeným časem akcí blízkým skutečnému výrobnímu času.

7. Výsledky experimentů

V této kapitole zhodnotíme výsledky metod diskutovaných v předchozích kapitolách na reálných datech, rozdělených na tři trénovací a testovací sady $C^t_{1,2,3}$ a $C^f_{1,2,3}$ dle kapitoly 3.4. Jelikož z důvodu omezeného množství dat nevyužíváme pevné oddělení validační množiny z trénovacích dat, pro všechny modely jsme využili krosvalidaci pro nalezení nejlepších parametrů a odhadu standardní odchylky finální predikce. Následně jsme provedli vyhodnocení modelů na $C^f_{1,2,3}$ s nejlepšími nalezenými parametry – modely byly natrénovány na celé trénovací sadě a byla spočtena střední hodnota ztrátové funkce *loss* na trénovací (TL) a testovací (FL) množině.

Hodnota ztrátové funkce $loss\left(t_{pred}\left(a\right), t\left(a\right)\right)$ nemusí být snadno interpretovatelná. Podívejme se tedy na přímočařejší interpretaci pomocí relativní chyby predikce. Závislost mezi poměrem zahlášeného a predikovaného času (dělíme menší z těchto dvou hodnot) a *loss* můžeme vidět na obrázku 4.1. Pro zpřesnění přidejme tabulky 7.1 s přesnými převody pro vybrané poměry časů a hodnoty *loss*.

Pro zpřehlednění výsledků si zaveďme zkratky pro jednotlivé typy modelů a jejich hyperparametry, viz tabulka 7.2. Vynecháme zde hyperparametry dostupné pro neuronové sítě, neboť vzhledem k jejich množství je vhodnější provést rozbor později pro každý hyperparametr zvlášť.

7.1 Hraniční hodnoty ztrátové funkce

Nejprve se zaměříme na zjištění limitních hodnot *loss*, které můžeme získat z libovolného modelu. Za horní odhad můžeme považovat predikci jediného přípravného a jednicového času pro celou trénovací množinu – vzhledem k neomezenosti chybové funkce *err* lze zjevně provést predikci dosahující libovolně vysoké hodnoty *loss*, nicméně jakýkoli smysluplný model by neměl být horší než tento odhad. Výsledky při použití SLR na celá vstupní data lze nalézt v tabulce 7.3.

Poznámka. Odhadované standardní odchylky *loss* v tabulce 7.3 nejspíše nejsou příliš přesné, neboť pro takto vysoké hodnoty *loss* by pro přesnější odhad bylo

Poměr časů	*loss*	*loss*	Poměr časů
1,1	0,009	0,05	1,251
1,2	0,033	0,1	1,372
1,3	0,069	0,2	1,564
1,5	0,164	0,3	1,729
1,75	0,313	0,4	1,882
2	0,480	0,5	2,028
2,5	0,840	0,75	2,377
3	1,207	1	2,718
4	1,922	1,5	3,403
5	2,590	2	4,113

Tabulka 7.1: Vztahy mezi ztrátovou funkcí a poměrem zahlášeného a predikovaného času

Název modelu či parametru	Zkratka
Standardní LR	SLR
Vážená LR	VLR
Iterativní LR	ILR
Kombinovaná LR	KLR
Optimální LR	OLR
Theil-Senův regresor	TSR
LDA clusterizace	LDA
Neuronová síť	NN
Slučování vzácných typů operací	SO
Detekce outlierů pomocí Z-skóre	ZS
Detekce outlierů pomocí Z-skóre s logaritmovanými $t(a)$	ZSL
Detekce outlierů pomocí DBSCAN	DB
Detekce outlierů pomocí DBSCAN s logaritmovanými $t(a)$	DBL
Dynamické určení t_b při neplatném výsledku LR	DP
Parametr α kombinované LR	α
Minimální počet akcí pro typ operace	MA
Relativní velikost dat označených jako outliery	RO

Tabulka 7.2: Seznam zkratek pro jednotlivé typy použitých modelů

Sada	TL	FL
Všechna data	$4{,}937_{\pm 0{,}026}$	-
C_1	$4{,}955_{\pm 0{,}042}$	$4{,}929_{\pm 0{,}054}$
C_2	$4{,}917_{\pm 0{,}087}$	$4{,}932_{\pm 0{,}109}$
C_3	$5{,}059_{\pm 0{,}015}$	$3{,}757_{\pm 0{,}018}$

Tabulka 7.3: Hodnota *loss* konstantní predikce t_b, t_a pro všechna vstupní data

třeba použít při krosvalidaci rozdělení na více než 5 množin.

Na testovací sadě C_3^f je výsledek výrazně lepší než všechny ostatní hodnoty. To je způsobeno tím, že v C_3^f se vyskytují pouze akce z operací, které byly prováděny často. Skutečný výrobní čas takových operací je obvykle blíže průměru než výrobní časy „speciálních“ operací. To je v souladu s tím, že výsledek na C_3^t je naopak mírně horší než na ostatních trénovacích množinách, neboť obsahuje všechny vzácné operace.

Opačným přístupem je nalezení co nejtěsnějšího dolního odhadu *loss* skutečných výrobních časů T, který tak nemůže žádný model překonat. K tomu použijeme nejlepší možnou predikci pro celou množinu $\mathcal{A}$ za pomoci OLR z kapitoly 4.2.2. Ohodnocení se následně provede standardně na trénovacích a testovacích množinách. Hodnoty optimálních predikcí můžeme nalézt v tabulce 7.4.

Povšimněme si, že hledáme nejlepší možnou predikci pro všechna data, nedostaneme tak nutně nejnižší hodnotu *loss* dosažitelnou pro podmnožiny $\mathcal{A}$, zejména pak pro testovací množiny. Provedení regrese pouze pro tyto podmnožiny by však nedávalo moc smysl, neboť v nich budou pro jednotlivé operace $V \in \mathcal{V}$ často nej-

Sada	TL	FL
Všechna data	$0{,}114_{\pm 0{,}001}$	-
C_1	$0{,}114_{\pm 0{,}001}$	$0{,}113_{\pm 0{,}009}$
C_2	$0{,}114_{\pm 0{,}002}$	$0{,}113_{\pm 0{,}004}$
C_3	$0{,}108_{\pm 0{,}001}$	$0{,}172_{\pm 0{,}008}$

Tabulka 7.4: Odhad hodnoty *loss* pro skutečné výrobní časy T

výše 2 akce [1]. Jednou akcí lze vždy vést predikci s kladným přípravným i jednicovým časem, Stejně tak i velkou částí dvojic akcí (pro 2 akce může vyjít záporný přípravný či jednicový čas). V těchto případech by tedy byla hodnota *loss* vždy 0. Celkový odhad optimální *loss* by tak byl výrazně podhodnocen.

Z tabulky 7.4 lze vyčíst, že šum ve vstupních datech je skutečně poměrně značný. Výrazně horší výsledek OLR na množině C_3^f může být překvapivý, ovšem jedná se o přímý důsledek pozorování z předchozího odstavce – v C_3^f se vyskytují pouze akce těch operací V, pro které je $|\mathcal{A}_V| \geq 9$. Nedochází tak k výraznému snížení průměrné *loss* přesnou predikcí pro operace s malým množstvím akcí. Lze předpokládat, že $loss \approx 0{,}172$ bude blízká skutečnému optimu pro zdrojovou průmyslovou výrobu. Tato hodnota odpovídá poměru zahlášeného a predikovaného času přibližně 1,5, zahlášené časy se tedy v průměru od skutečného výrobního času odchylují o 50%.

7.2 LR modely

7.2.1 Modely založené na variantách SLR

Prvním typem diskutovaných modelů v práci byly modely založené na klasické LR. Vzhledem k vyšší rychlosti výpočtu (viz tabulka 7.5) bylo možné vyzkoušet všechny relevantní kombinace parametrů (tzv. grid search), především pak vliv detekce outlierů na kvalitu predikce. Pro každý typ testované LR jsme porovnali její chování bez modifikace dat a při použití filtrací SO, ZS(L) a DB(L) na vstupní data.

Výsledky pro SLR můžeme nalézt v tabulce 7.6, pro VLR v tabulce 7.7, pro ILR v tabulce 7.8 a pro KLR v tabulce 7.9. Pro grid search byly použity následující varianty hyperparametrů, pokud daný hyperparametr je pro testovaný model relevantní:

- **DP**: true; false (pro false se při neplatné regresi použije $t_b = 0$, pro true dynamický průsečík diskutovaný v sekci 4.1.1)
- **MA**: 5; 20; 100
- **RO**: 0,002; 0,01; 0,05
- $\boldsymbol{\alpha}$: 0,25; 0,5; 0,75 (výsledky v tabulce 7.9 ukázaly, že hledání hodnot blíže 0,5 by mohlo být vhodnější)

[1] zejména v C_1^f a C_2^f vzhledem k jejich velikosti a obsahu, vyjdeme-li z grafu rozložení počtů akcí jednotlivých operací 3.8

Model	Přibližný čas běhu
SO	< 1 s
ZS(L)	3 s
DB(L)	2 min
SLR	3 s
VLR	3 s
ILR	5 s
KLR	5 s
OR	10 min
TSR	1 min
LDA	1 iterace 1,5 min, celkem cca 30 min
NN	1 epocha cca 2,5 min, celkem 30-40 min dle počtu

Tabulka 7.5: Časové nároky jednotlivých typů modelů na použitém zdroji (viz příloha A.2)

Pro trénovací chyby neuvádíme ve výsledcích LR standardní odchylku, neboť je ve všech případech zanedbatelná – nikdy nepřekročila hodnotu 0,003. Rovněž neuvádíme výsledky regresorů pouze s SO. Toto slučování má vliv pouze na malou část akcí, proto se na celkových výsledcích prakticky neprojeví (rozdíl je obvykle o řád nižší než standardní odchylka).

Poznámka. Jelikož je obsah trénovacích a testovacích množin shodný pro všechny modely, zjištěné TL a FL pro jednotlivé modely by mezi sebou měly výrazně korelovat. Proto výsledky dvou různých modelů na určité sadě dat lišící se o standardní odchylku ve skutečnosti určují téměř jednoznačně lepší model, především v situaci, kdy oba modely používají podobnou myšlenku.

Z výsledků lze provést řadu pozorování:

- SLR dosahuje stabilně nejhorších výsledků jak na trénovacích, tak na testovacích množinách. Toto není překvapivé, jelikož SLR optimalizuje funkci výrazně se lišící od *loss*.
- TL všech LR se poměrně výrazně přibližuje optimální predikci z tabulky 7.4, především při nízkém RO[2]. Modely založené na LR tedy mají pro náš problém tendenci se přeučovat. Je to způsobeno zejména tím, že pro většinu operací je počet jejich akcí $|A_V|$ velmi nízký (viz obrázek 3.7) a tyto modely nejsou schopny nalézt provázanost mezi nimi.
- Na C_2^f dosahují LR modely výrazně horších výsledků, neboť musejí vždy použít záložní predikci dle typu operace. Pro C_3^f naopak výsledky nejsou vzdálené od optimální predikce. Ukazuje se tak, že pro predikci známých operací by měla být LR (v našem případě KLR) dobrým kompromisem mezi rychlostí a kvalitou predikce.

[2]Jelikož vyfiltrované outliery nejsou použité pro učení, dá se na ních očekávat podobná úspěšnost jako na testovací množině. Snížení RO sníží počet vyfiltrovaných akcí, čímž omezí jejich vliv na TL

Sada	Model	TL	FL	Nejlepší parametry
C_1	SLR	0,142	$0{,}605_{\pm 0{,}005}$	DP=false
	SLR+SO+ZS	0,181	$0{,}467_{\pm 0{,}008}$	DP=true, MA=5, RO=0,05
	SLR+SO+ZSL	0,190	$0{,}475_{\pm 0{,}008}$	DP=true, MA=20, RO=0,05
	SLR+SO+DB	0,174	$0{,}503_{\pm 0{,}011}$	DP=true, MA=5, RO=0,05
	SLR+SO+DBL	0,159	$0{,}480_{\pm 0{,}004}$	DP=true, MA=5, RO=0,05
C_2	SLR	0,157	$1{,}248_{\pm 0{,}029}$	DP=true
	SLR+SO+ZS	0,180	$0{,}929_{\pm 0{,}024}$	DP=true, MA=5, RO=0,05
	SLR+SO+ZSL	0,189	$0{,}929_{\pm 0{,}024}$	DP=true, MA=5, RO=0,05
	SLR+SO+DB	0,177	$1{,}019_{\pm 0{,}023}$	DP=true, MA=5, RO=0,05
	SLR+SO+DBL	0,159	$1{,}014_{\pm 0{,}029}$	DP=true, MA=5, RO=0,05
C_3	SLR	0,139	$0{,}254_{\pm 0{,}005}$	DP=false
	SLR+SO+ZS	0,138	$0{,}252_{\pm 0{,}004}$	DP=false, MA=5, RO=0,002
	SLR+SO+ZSL	0,139	$0{,}254_{\pm 0{,}005}$	DP=false, MA=5, RO=0,002
	SLR+SO+DB	0,143	$0{,}253_{\pm 0{,}006}$	DP=false, MA=5, RO=0,01
	SLR+SO+DBL	0,143	$0{,}252_{\pm 0{,}006}$	DP=false, MA=5, RO=0,01

Tabulka 7.6: Výsledky experimentů pro standardní LR

Sada	Model	TL	FL	Nejlepší parametry
C_1	VLR	0,131	$0{,}351_{\pm 0{,}008}$	DP=true
	VLR+SO+ZS	0,137	$0{,}346_{\pm 0{,}007}$	RO=0,01, MA=20, DP=true
	VLR+SO+ZSL	0,133	$0{,}342_{\pm 0{,}007}$	RO=0,01, MA=20, DP=true
	VLR+SO+DB	0,140	$0{,}350_{\pm 0{,}007}$	RO=0,01, MA=20, DP=true
	VLR+SO+DBL	0,141	$0{,}339_{\pm 0{,}008}$	RO=0,05, MA=5, DP=true
C_2	VLR	0,133	$0{,}937_{\pm 0{,}039}$	DP=true
	VLR+SO+ZS	0,140	$0{,}911_{\pm 0{,}039}$	RO=0,01, MA=100, DP=true
	VLR+SO+ZSL	0,135	$0{,}899_{\pm 0{,}036}$	RO=0,01, MA=100, DP=true
	VLR+SO+DB	0,143	$0{,}915_{\pm 0{,}034}$	RO=0,01, MA=5, DP=true
	VLR+SO+DBL	0,142	$0{,}881_{\pm 0{,}035}$	RO=0,05, MA=5, DP=true
C_3	VLR	0,126	$0{,}236_{\pm 0{,}005}$	DP=true
	VLR+SO+ZS	0,129	$0{,}237_{\pm 0{,}005}$	RO=0,002, MA=100, DP=true
	VLR+SO+ZSL	0,128	$0{,}234_{\pm 0{,}004}$	RO=0,01, MA=5, DP=true
	VLR+SO+DB	0,135	$0{,}239_{\pm 0{,}005}$	RO=0,01, MA=5, DP=true
	VLR+SO+DBL	0,127	$0{,}235_{\pm 0{,}004}$	RO=0,01, MA=5, DP=true

Tabulka 7.7: Výsledky experimentů pro váženou LR

Sada	Model	TL	FL	Nejlepší parametry
C_1	ILR	0,134	$0{,}418_{\pm 0{,}007}$	DP=true
	ILR+SO+ZS	0,170	$0{,}402_{\pm 0{,}008}$	RO=0,05, MA=5, DP=true
	ILR+SO+ZSL	0,137	$0{,}395_{\pm 0{,}007}$	RO=0,01, MA=20, DP=true
	ILR+SO+DB	0,160	$0{,}405_{\pm 0{,}008}$	RO=0,05, MA=20, DP=true
	ILR+SO+DBL	0,147	$0{,}395_{\pm 0{,}007}$	RO=0,05, MA=20, DP=true
C_2	ILR	0,137	$0{,}932_{\pm 0{,}030}$	DP=true
	ILR+SO+ZS	0,170	$0{,}829_{\pm 0{,}027}$	RO=0,05, MA=5, DP=true
	ILR+SO+ZSL	0,178	$0{,}824_{\pm 0{,}027}$	RO=0,05, MA=5, DP=true
	ILR+SO+DB	0,162	$0{,}854_{\pm 0{,}024}$	RO=0,05, MA=5, DP=true
	ILR+SO+DBL	0,148	$0{,}885_{\pm 0{,}026}$	RO=0,05, MA=5, DP=true
C_3	ILR	0,130	$0{,}245_{\pm 0{,}005}$	DP=true
	ILR+SO+ZS	0,130	$0{,}242_{\pm 0{,}004}$	RO=0,002, MA=20, DP=true
	ILR+SO+ZSL	0,130	$0{,}243_{\pm 0{,}005}$	RO=0,002, MA=20, DP=true
	ILR+SO+DB	0,130	$0{,}244_{\pm 0{,}005}$	RO=0,002, MA=20, DP=true
	ILR+SO+DBL	0,132	$0{,}241_{\pm 0{,}005}$	RO=0,01, MA=20, DP=true

Tabulka 7.8: Výsledky experimentů pro iterovanou LR

Sada	Model	TL	FL	Nejlepší parametry
C_1	KLR	0,121	$0{,}328_{\pm 0{,}005}$	DP=true, α=0,5
	KLR+SO+ZS	0,126	$0{,}325_{\pm 0{,}006}$	RO=0,01, MA=20, DP=true, α=0,5
	KLR+SO+ZSL	0,124	$0{,}321_{\pm 0{,}005}$	RO=0,01, MA=20, DP=true, α=0,5
	KLR+SO+DB	0,128	$0{,}332_{\pm 0{,}006}$	RO=0,01, MA=20, DP=true, α=0,5
	KLR+SO+DBL	0,123	$0{,}323_{\pm 0{,}005}$	RO=0,01, MA=20, DP=true, α=0,5
C_2	KLR	0,123	$0{,}753_{\pm 0{,}026}$	DP=true, α=0,5
	KLR+SO+ZS	0,167	$0{,}753_{\pm 0{,}037}$	RO=0,05, MA=100, DP=true, α=0,5
	KLR+SO+ZSL	0,126	$0{,}743_{\pm 0{,}033}$	RO=0,01, MA=100, DP=true, α=0,5
	KLR+SO+DB	0,131	$0{,}754_{\pm 0{,}026}$	RO=0,01, MA=20, DP=true, α=0,5
	KLR+SO+DBL	0,135	$0{,}737_{\pm 0{,}025}$	RO=0,05, MA=20, DP=true, α=0,5
C_3	KLR	0,117	$0{,}224_{\pm 0{,}005}$	DP=true, α=0,5
	KLR+SO+ZS	0,119	$0{,}223_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true, α=0,5
	KLR+SO+ZSL	0,117	$0{,}223_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true, α=0,5
	KLR+SO+DB	0,118	$0{,}224_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true, α=0,5
	KLR+SO+DBL	0,119	$0{,}222_{\pm 0{,}005}$	RO=0,01, MA=5, DP=true, α=0,5

Tabulka 7.9: Výsledky experimentů pro kombinovanou LR

- Používání DP namísto pouhého $t_b = 0$ téměř vždy zlepšuje výsledky. Jedinou výjimkou je SLR, zejména pro C_3^f – to je ovšem nejspíše rovněž způsobené problematickou chybovou funkcí z prvního pozorování.

- Přestože je algoritmus detekce outlierů pomocí metody DBSCAN obecnější než Z-skóre a měl by tak dávat lepší výsledky, je výsledek většinou opačný. Toto je pravděpodobně způsobeno příliš nehomogenním rozložením dat (některé typy operací obsahují mnoho akcí se stejnými zahlášenými hodnotami, což zhoršuje správnou detekci jádrových bodů) a větší tendenci Z-skóre označovat extrémnější hodnoty za outliery bez ohledu na vzácnost (viz obrázek 3.6), což je zejména pro jednodušší LR modely vhodnější.

- Použití logaritmu zahlášených časů při detekci outlierů je obvykle mírně lepší, byť toto pozorování je na hranici statistické chyby. Jelikož je však jeho použití na rozdíl od používání absolutních rozdílů časů v souladu s err, bude to pro komplexnější modely preferovaná varianta.

- KLR dosahuje pro všechny testovací množiny nejlepších výsledků. Zkombinování výhod VLR a ILR se tedy ukázalo jako dobrý přístup.

- Nejlepší hodnota parametru α pro KLR ze zkoušených variant vždy vyšla jako 0,5. Podle všeho je tak optimální poměr vážené a iterované verze LR blízký rovnovážnému. Vyzkoušení dalších hodnot blízkých 0,5 by mohlo optimální poměr ještě mírně zpřesnit – vzhledem k podobným výsledkům pro C_1 s $\alpha = 0{,}25$ a pro C_2 s $\alpha = 0{,}75$ bude optimální α závislé na struktuře testovacích dat.

7.2.2 Theil-Senův regresor

Jak můžeme vidět v tabulce 7.5, trénování Theil-Senova regresoru je o řád pomalejší než předcházející regresory, zejména kvůli nutnosti zpracovat směrnice pro všechny dvojice vstupních bodů[3]. Stále je však dostatečně rychlé pro grid search všech kombinací hyperparametrů. Výsledky experimentů nalezneme v tabulce 7.10.

Klíčovým pozorováním je kompletní eliminace vlivu filtrace outlierů na výsledky, při použití v reálném provozu lze tedy doporučit přídavnou filtraci vynechat. Zdá se, že zejména DB výsledky dokonce mírně zhoršuje, byť rozdíly jsou pod hranicí statistické chyby. Možným vysvětlením může být tendence filtrací chybně odstraňovat data extrémnějších operací, TR však i pro takové operace dokáže provést dobrou predikci.

Bohužel, pro C_1^f a C_2^f dosahuje TR výsledků pouze srovnatelných s KLR, pro časté operace C_3^f je dokonce o trochu horší. Tento výsledek je nejspíše způsoben tím, že TR očekává přibližně stejný počet zahlášených časů vyšších a nižších než skutečný výrobní čas. To by sice pro větší počet akcí měla být dle pozorování v kapitole 3.2 pravda, zejména pro operace s malým počtem akcí v trénovací množině je však pravděpodobnost odchýlení mediánu poměrně vysoká. Pro C_3^f lze tedy vzhledem k výsledkům i časové náročnosti doporučit spíše filtraci outlierů s nerobustní variantou LR než přímé využití TR.

[3]Byla využita implementace z knihovny `sklearn`, která využívá rychlejší algoritmus než generování všech $\binom{n}{2}$ směrnic s následným hledáním jejich mediánu.

Sada	Model	TL	FL	Nejlepší parametry
C_1	TR	0,135	$0{,}325_{\pm 0{,}006}$	DP=true
	TR+SO	0,135	$0{,}325_{\pm 0{,}006}$	MA=20, DP=true
	TR+SO+ZS	0,136	$0{,}326_{\pm 0{,}007}$	RO=0,002, MA=20, DP=true
	TR+SO+ZSL	0,135	$0{,}325_{\pm 0{,}006}$	RO=0,002, MA=20, DP=true
	TR+SO+DB	0,141	$0{,}329_{\pm 0{,}007}$	RO=0,01, MA=20, DP=true
	TR+SO+DBL	0,139	$0{,}328_{\pm 0{,}006}$	RO=0,01, MA=20, DP=true
C_2	TR	0,137	$0{,}740_{\pm 0{,}030}$	DP=true
	TR+SO	0,137	$0{,}748_{\pm 0{,}035}$	MA=100, DP=true
	TR+SO+ZS	0,180	$0{,}749_{\pm 0{,}036}$	RO=0,05, MA=100, DP=true
	TR+SO+ZSL	0,185	$0{,}749_{\pm 0{,}035}$	RO=0,05, MA=100, DP=true
	TR+SO+DB	0,168	$0{,}749_{\pm 0{,}035}$	RO=0,05, MA=100, DP=true
	TR+SO+DBL	0,141	$0{,}750_{\pm 0{,}035}$	RO=0,01, MA=100, DP=true
C_3	TR	0,130	$0{,}242_{\pm 0{,}005}$	DP=true
	TR+SO	0,130	$0{,}242_{\pm 0{,}005}$	MA=5, DP=true
	TR+SO+ZS	0,131	$0{,}241_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true
	TR+SO+ZSL	0,131	$0{,}242_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true
	TR+SO+DB	0,131	$0{,}242_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true
	TR+SO+DBL	0,130	$0{,}242_{\pm 0{,}005}$	RO=0,002, MA=5, DP=true

Tabulka 7.10: Výsledky experimentů pro Theil-Senův regresor

Sada	Model	TL	FL	Nejlepší parametry
C_1	OLR	0,112	$0{,}388_{\pm 0{,}008}$	
	OLR+SO+ZSL	0,121	$0{,}384_{\pm 0{,}007}$	RO=0,01, MA=20
C_2	OLR	0,114	$0{,}757_{\pm 0{,}031}$	
	OLR+SO+ZSL	0,123	$0{,}726_{\pm 0{,}027}$	RO=0,01, MA=20
C_3	OLR	0,106	$0{,}235_{\pm 0{,}004}$	
	OLR+SO+ZSL	0,108	$0{,}237_{\pm 0{,}006}$	RO=0,002, MA=5

Tabulka 7.11: Výsledky experimentů pro optimální LR

7.2.3 Optimální LR

Primárním účelem OLR bylo zjistit nejlepší dosažitelný odhad výrobních časů T, její využití pro skutečnou predikci je tedy provedeno především pro zkompletování všech využitých LR. Výsledky můžeme najít v tabulce 7.11. Na rozdíl od ostatních LR jsme vzhledem k vyšší časové náročnosti nevyužili grid search všech filtrací a hyperparametrů, ale pouze ZSL s vyzkoušením všech RO.

Výsledky jsou překvapivě dobré, zejména pro C_2^f se ZSL. Pro tuto testovací sadu dokonce OLR mírně překonal výsledky KLR. Pravděpodobným zdůvodněním je fakt, že všechny použité LR musejí pro C_2^f používat záložní predikci dle typu provedené operace. Většina typů operací však obsahuje stovky až tisíce akcí, modely by tak měly pro takto velké množiny predikovat přípravný a jednicový čas blízký optimálnímu vzhledem k chyové funkci odpovídajícího modelu. Jelikož OLR je jediný LR model, pro který je chybová funkce identická s *loss*, jeho

Sada	Model	TL	FL
C_1	LDA (10 clusterů) + ZSL	2,113	2,061
	LDA (30 clusterů)	1,675	1,636
	LDA (30 clusterů) + ZSL	1,590	1,560
C_2	LDA (10 clusterů) + ZSL	2,096	2,173
	LDA (30 clusterů)	1,627	1,680
	LDA (30 clusterů) + ZSL	1,742	1,806
C_3	LDA (10 clusterů) + ZSL	1,950	1,722
	LDA (30 clusterů)	1,645	1,401
	LDA (30 clusterů) + ZSL	1,619	1,397

Tabulka 7.12: Výsledky experimentů pro LDA

predikce bývají mírně lepší.

Je pravděpodobné, že využitím některých vylepšení z ostatních LR, především pro operace V s malým $|A_V|$, by bylo možné výsledky OLR pro testovací množiny ještě vylepšit. Vzhledem k časovým nárokům na trénování tohoto modelu jsme však věnovali více času vylepšením pro NN, jejíž výsledky výrazně dominují predikce LR modelů.

7.3 algoritmus LDA

Hledání podobností mezi operacemi pomocí unsupervised clusterizace a následné provedení LR (konkrétně KLR) pro jednotlivé clustery je netradičním řešením, které by pro určité typy dat mohlo dávat dobré výsledky. Vzhledem k velikosti clusterů lze očekávat, že standardní odchylky KLR jednotlivých clusterů budou blízké skutečným, na rozíl od předchozích modelů tedy lze LDA použít i pro predikci očekávané chyby predikovaných časů.

Samotná clusterizace dosahovala dobrých výsledků, jak můžeme vidět na obrázku 5.1. Bohužel se však ukázalo, že rozdělení popisů operací $V \in \mathcal{V}$ na jednotlivá slova nereflektuje jejich vliv na $T(V)$ – v popisech zdrojových dat má výrazný vliv kontext (např. „Řezání tyče“ a „Začištění tyče po řezání“ jsou zcela odlišné operace, nicméně LDA je přiřadí do stejného clusteru). Většina popisů je navíc v češtině, shodná slova s jinou koncovkou tak LDA bude považovat za různá. Vzhledem k anonymizaci dat je navíc problematické provedení pokročilejší jazykové analýzy pro zredukování tohoto problému.

Výsledky provedených experimentů můžeme nalézt v tabulce 7.12. Vzhledem k výrazně horším výsledkům jsme neprováděli hledání optimálních hyperparametrů, byly ohodnoceny jen různé počty clusterů a vliv ZSL s RO=0,01, pro zbylé hyperparametry byly ponechány defaultní hodnoty. Ze stejného důvodu jsme vynechali analýzu úspěšnosti predikce *loss*, byť ji tento model podporuje.

7.4 Modely založené na predikci pomocí neuronové sítě

V poslední části práce jsme se zabývali modely založenými na umělých neuronových sítích a jejich návrhem pro co nejlepší využití informací o jednotlivých operacích, diskutovaným v sekci 6.1. Vzhledem k velkému počtu hyperparametrů výsledného modelu NN a dlouhého času učení není možné provést v tomto případě grid search. Namísto toho budeme hledat optimální nastavení jednotlivých parametrů postupně, v pořadí dle očekávaného vlivu na výsledky. Tímto způsobem sice pravděpodobně nenalezneme optimální řešení[4], nicméně by od něj nemělo být příliš vzdálené. Jednotlivé atributy také podporují nastavení vlastních hyperparametrů použitých při jejich embeddingu, až na výjimky jsme však vždy použili hodnotu společnou pro celou síť kvůli neprůkaznému vlivu na výsledky.

Použití krosvalidace vyžaduje přibližně 2-3 hodiny k otestování jedné kombinace parametrů, dle tabulky 7.5. I s postupným řešením jednotlivých hyperparametrů tak jejich optimalizace vyžaduje přibližně týden výpočetního času. Pro zrychlení učení se tedy zaměříme nejprve na testovací sadu C_2, která je pro nás nejzajímavější vzhledem k slabým výsledkům ostatních modelů. Všechny hyperparametry optimalizované na této sadě a zvolené hodnoty můžeme nalézt v tabulce 7.13. Pro zbylé sady vyjdeme z optimálních hyperparametrů pro C_2 a provedeme hledání pouze pro ty hyperparametry, pro které lze očekávat statisticky významné změny.

Poznámka. Implementace NN podporuje i několik parametrů, které v tabulce 7.13 nejsou zmíněné. Jejich hodnoty však buďto nejsou zkoumané, nebo jde o pomocné atributy upravující chování sítě. Zběžný popis takových parametrů lze nalézt v příloze A.2.

Parametr	Hodnoty	Informace o parametru
batch size	128, 256, **512**, 1024	Počet akcí v jednom bloku dat. Kvůli paměťovým limitům nebylo možné vyzkoušet hodnoty větší než 1024.
epochs	8, **10**, 12, 15	Počet průchodů celé trénovací množiny během učení. Pro 12 epoch vycházela podobná *loss* na validačních sadách, ale nižší *loss* na trénovací sadě, tj. docházelo k většímu přeučování.
dense size	32, **48**, 64, 128	Počet neuronů ve všech dense vrstvách, není-li pro určitou vrstvu specifikována jiná hodnota.
multi scale rate	1, 1/2, 1/4, **1/8**	Škálovací parametr pro určení α ze sekce 6.1.5. Jako α použijeme maximální počet materiálů produktů přenásobený tímto parametrem.

Pokračování na další straně

[4]Vytvořený kód pro validaci umožňuje provést grid search i podmnožiny hyperparametrů, což je vhodné pro některé výrazně provázané hyperparametry jako například frekvence a koeficient snižování learning rate

Parametr	Hodnoty	Informace o parametru
BOW / EOW	**true**, false	Zda mají být při embeddingu řetězců použity speciální znaky BOW a EOW.
embedding size	32, 48, **64**, 128	Počet neuronů pro embedding každého znaku řetězců.
prediction layers	1, **2**, 3	Počet dense vrstev pro predikci výrobního času.
prediction layer size	64, 128, **256**	Počet neuronů ve vrstvách „prediction layers“ (parametry byly optimalizovány současně).
error layers	**1**, 2, 3	Počet dense vrstev pro predikci chyby.
error layer size	64, 128, **256**	Počet neuronů ve vrstvách „error layers“ (parametry byly optimalizovány současně).
RNN type	**LSTM**, GRU	Použitý typ RNN dle 6.1.4. Ukázalo se, že pro délku jednotlivých popisů operací a produktů více vyhovuje dlouhodobější „paměť“ LSTM.
dense size výšky, šířky a hloubky	8, 16, **32**, 48	Speciální velikost dense vrstev pro výšku, šířku a hloubku materiálů. Byly zkoušeny pouze kombinace se všemi třemi tři hodnotami nastavenými shodně. Vzhledem k jednoduchosti vstupu se dala předpokládat výhodnost výrazného zmenšení, to se nicméně příliš nepotvrdilo.
LR decay time	200, **300**, 500	Frekvence snižování learning rate η, tj počet operací, při kterém je η přenásobeno LR decay ratio.
LR decay ratio	0,5, 0,6, **0,7**, 0,8	Poměr pro snížení learning rate každých LR decay time iterací. Optimalizováno v kombinaci s předchozím hyperparametrem.
t_a activation	**softplus^2**, dle variant v 6.2	Aktivační funkce pro neuron predikující t_a.
t_b activation	**softplus**, dle variant v 6.2	Aktivační funkce pro neuron predikující t_b. Trochu překvapivě se softplus^2 ukázala jako horší.
dropout	0,01, 0,02, 0,05, **0,1**, 0,2, 0,5	Velikost dropoutu. Pro méně než 0,05 docházelo k přeučování, pro více než 0,1 by naopak dense vrstvy musely být větší kvůli podtrénování.
λ	2, 5, **10**, 15	Hodnota hyperparametru λ pro predikci *loss* dle sekce 6.4.
epochs	8, 10, **12**, 15	Aktualizace optimálního počtu epoch, došlo-li ke změně při úpravách hyperparametrů.

Tabulka 7.13: Hyperparametry NN s testovanými variantami při hledání jejich optimální kombinace, s vyznačením nejlepší nalezené volby

Sada	Model	TL	FL
C_1	ZNN + PL	$0{,}202_{\pm 0{,}002}$	$0{,}269_{\pm 0{,}005}$
	ZNN + MAT	$0{,}190_{\pm 0{,}001}$	$0{,}252_{\pm 0{,}003}$
	ZNN + PL + MAT	$0{,}191_{\pm 0{,}001}$	$0{,}251_{\pm 0{,}002}$
C_2	ZNN + PL[6]	$0{,}217_{\pm 0{,}003}$	$0{,}428_{\pm 0{,}018}$
	ZNN + MAT	$0{,}196_{\pm 0{,}001}$	$0{,}380_{\pm 0{,}014}$
	ZNN + PL + MAT	$0{,}197_{\pm 0{,}001}$	$0{,}380_{\pm 0{,}014}$
C_3	ZNN + PL	$0{,}210_{\pm 0{,}001}$	$0{,}227_{\pm 0{,}003}$
	ZNN + MAT	$0{,}189_{\pm 0{,}002}$	$0{,}217_{\pm 0{,}002}$
	ZNN + PL + MAT	$0{,}189_{\pm 0{,}001}$	$0{,}216_{\pm 0{,}002}$

Tabulka 7.14: Výsledky experimentů pro neuronové sítě

Pro určení optimálních parametrů pro sady C_1 a C_3 byly vyzkoušeny změny velikostí jednotlivých vrstev a úpravy počtu epoch. Jedinými změnami, které na validačních množinách zlepšily výsledky, však bylo zvýšení dense size na 64 a navýšení počtu epoch na 15. Problém přeučování byl na rozdíl od C_2 výrazně nižší, přidání stupňů volnosti a doby učení proto pomohlo stabilně snižovat průměrnou hodnotu TL i FL přibližně o 0,01.

Kromě ohodnocení celé sítě by nás také zajímal vliv jejích pokročilých částí na úspěšnost, zda se tedy zvýšení výpočetních nároků vyplatí. Rozdělme si síť dle obrázku 6.1 na 3 podčásti:

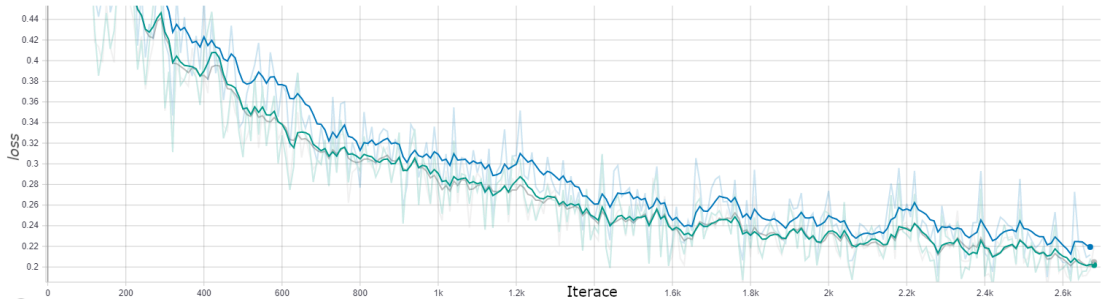
- **MAT**: Část využitá pro embedding informací zdrojových materiálů jednotlivých produktů. Jelikož tato část přidává dodatečnou dimenzi vstupu kvůli variabilnímu počtu materiálů, průběh učení je tak zpomalen. Největším problémem jsou řetězcová data, proto nakonec pro učení nebyl atribut „Název produktu“ jednotlivých materiálů použitý.
- **PL**: Část využitá pro predikci *loss*, definovaná v 6.4. Kromě výhody existence této predikce je ve zdrojovém článku diskutován pozitivní vliv na učení celé sítě v situaci, kdy různé typy dat mají různé rozptyly – chtěli bychom tedy zjistit, zda se to týká i našeho problému.
- **ZNN**: Základní část neuronové sítě, tj. celý zbytek grafu bez MAT a PL. Není-li PL k dispozici, jako ztrátová funkce pro trénování se použije *loss*.

Provedli jsme ohodnocení pro všechny kombinace kromě samostatné ZNN[5], výsledky lze nalézt v tabulce 7.14. Hlavními důsledky těchto výsledků jsou:

- Překvapivě vysoký pozitivní vliv dostupných informací o materiálech, a to pro TL i FL. V grafu 7.1 můžeme vidět, že bez materiálu se po celou dobu učení stabilně pohybuje hodnota *loss* v rozmezí o 0,01–0,02 horším, než pro zbylé 2 typy sítí.

[5]Hodnocení samostatné ZNN jsme vynechali kvůli nulové přidané informační hodnotě – dle výsledků lze očekávat podobné hodnoty jako pro ZNN + PL, ovšem s nedostupnou predikcí *loss*.

[6]Použito pouze 10 epoch, rychlejší přeučování kvůli chybějícím materiálům dávalo pro 12 epoch horší výsledky.



Obrázek 7.1: Průběh učení na C_2 pro varianty sítí ZNN + PL (šedá), ZNN + MAT (modrá) a ZNN + PL + MAT (zelená). Vygenerováno s pomocí Tensorboard s koeficientem vyhlazování 0,8.

- PL má nulový či zanedbatelný vliv na kvalitu modelu – nebyla prováděna samostatná optimalizace hyperparametrů sítě ZNN + MAT, je tedy možné, že by taková varianta dosáhla i mírně lepších výsledků.
- Model založený na NN se je schopen naučit výrazně lepší predikci výrobních časů pro vzácné či zcela neznámé operace.
- Pro náhodnou množinu akcí C_1^f odpovídá střední odchylka od skutečného výrobního času přibližně 65%, pro zcela neznámou sadu akcí C_2^f pak přibližně 85%. Takové hodnoty jsou poměrně dobré, vezme-li v úvahu rozsahy validních výrobních časů a rozptyly zahlášených časů, jejichž průměr jsme v sekci 7.1 odhadli na 50%.
- Z grafu 7.2 můžeme vyčíst, že rozložení *loss* všech NN modelů je hladké a blízké exponenciálnímu lze tedy toto chování očekávat a využít i při reálném nasazení [7].

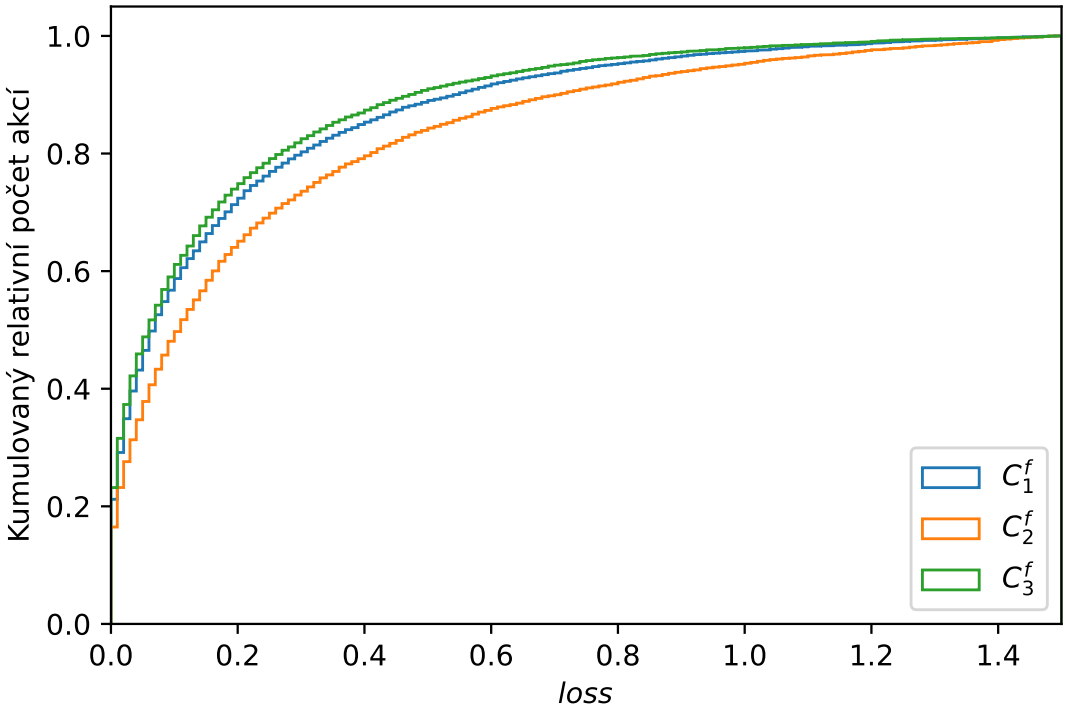
7.4.1 Predikce očekávané hodnoty loss

Pro další diskuzi již budeme používat pouze variantu ZNN + PL + MAT, neboť kvalitou dominuje variantu bez MAT a má k dispozici predikci *loss*. Zbývá se zaměřit na ohodnocení schopnosti predikce očekávané chyby, což je významná výhoda použitých NN modelů. Přímočarým řešením je zanalyzování přesnosti těchto predikcí na akcích odpovídajících testovacích množin.

Jelikož je střední hodnota predikcí *loss* vždy založena na trénovací množině, pro testované akce budou modely obvykle podhodnocovat skutečnou chybu. Pomocí krosvalidace však známe očekávaný poměr mezi trénovací a validační chybou pro jednotlivé typy vstupních dat, můžeme tak za předpokladu znalosti druhu dotazované operace[8] predikovanou chybu odpovídajícím poměrem přenásobit a výsledek použít jako predikci.

[7] Varianty LR naipříklad mají tendenci vytvářet různé skoky, především na poměrech odpovídajících číslům jako jsou $1, 2, \frac{3}{2}, \frac{4}{3}$ kvůli racionálním závislostem mezi souřadnicemi bodů jednotlivých akcí.

[8] Pokud neznáme přímo její zdroj (např. často prováděná operace, zcela neznámá operace apod.), lze použitý NN model rozšířit o počítadlo výskytů jednotlivých operací ve vstupních datech. Při krosvalidaci poté zjistíme očekávané poměry pro operace v závislosti na počtu n_V jejich akcí v trénovacích datech a získáme tak pro model pevnou mapovací funkci $r : \mathbb{N} \to \mathbb{R}$



Obrázek 7.2: Distribuce hodnot *loss* NN modelů na jednotlivých testovacích sadách.

Tyto poměry dostaneme tak, že po natrénování provedeme na odpovídajících validačních množinách predikci *loss*. Poměr skutečné *loss* proti průměrné predikci *loss* pak bude naším hledaným poměrem. Povšimněme si, že nelze s jistotou jednoduše použít poměr *loss* na validační a trénovací množině, neboť NN se může naučit nadprůměrnou predikci *loss* pro pro ni málo známé operace. Toto pozorování se do určité míry potvrdilo na několika manuálně provedených pokusech, například pro akce z C_2^f vycházela střední predikce *loss* dokonce téměř o 0,04 vyšší než pro akce z C_2^t pro tentýž model.

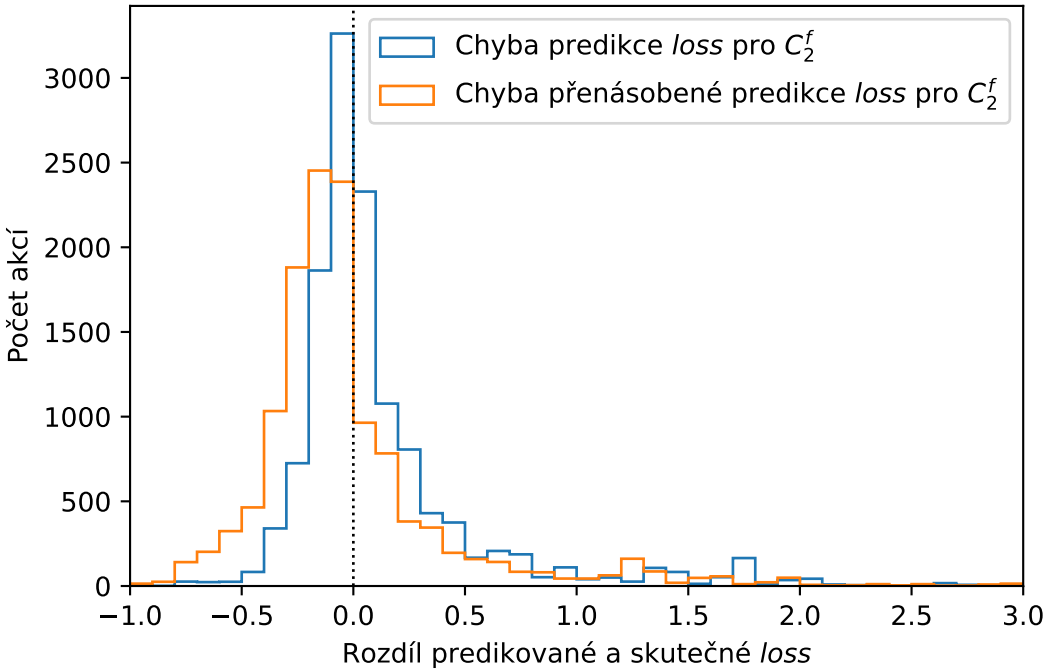
Analýzu výsledků predikcí *loss* pro jednotlivé testovací sady můžeme nalézt v tabulce 7.15, kde S značí originální predikci a P přeškálovanou. Důležitým ukazatelem je rozložení dle vzdálenosti predikce od skutečné *loss* (absolutní chybu a zejména pak standardní odchylku výrazně zvyšují outliery v testovací sadě), kdy i pro sady C_1^f a C_2^f s neznámými daty bylo 44% procent predikcí s přesností lepší než 0,1 a přibližně dvě třetiny predikcí s lepší než 0,2. Na C_3^f pak byly výsledky dle očekávání přesnější – je to tak možný argument pro reálné využití NN i pro takový typ produkce, byť lze na C_3^t zkonstruovat varianty predikcí *loss* i pro ostatní modely vzhledem ke garanci většího počtu dostupných akcí ke každé operaci.

Povšimněme si, že použití přeškálování predikcí dle popsaného odhadu sice skutečně téměř odstranilo střední chybu predikce, nicméně výrazně zhoršilo vý-

přiřazující každé operaci V očekávaný poměr $r(n_V)$, kterým budeme přenásobovat odpověď pro dotazovanou operaci V. Pro zpracování výsledků jsme však toto rozšíření neprovedli – u C_2 je pro všechny operace $n_V = 0$, u C_3 je vždy $n_V \geq 5$ a zpřesnění pomocí r tak bude zanedbatelné. Pouze pro C_1 by potenciálně mohlo dojít ke změně kvality, v práci se však tímto rozšířením nezabýváme dostatečně na to, aby ji bylo možné predikovat bez provedení experimentů.

Metrika	C_1^f		C_2^f		C_3^f	
	S	P	S	P	S	P
Střední chyba	0,047	0,000	0,148	-0,001	0,021	-0,001
Střední absolutní chyba	0,217	0,234	0,274	0,318	0,121	0,127
Standardní odchylka	0,522	0,522	0,599	0,588	0,260	0,261
Akcí s chybou v ±0,1	44,0%	36,6%	43,5%	26,1%	64,3%	60,5%
Akcí s chybou v ±0,2	71,1%	65,7%	66,4%	51,2%	86,7%	84,7%
Akcí s chybou v ±0,5	92,2%	92,1%	87,9%	84,8%	96,9%	96,9%
Akcí s chybou v ±1,0	96,9%	97,1%	94,1%	94,2%	99,3%	99,3%

Tabulka 7.15: Analýza výsledků predikce očekávané hodnoty *loss*



Obrázek 7.3: Rozložení chyb predikcí *loss* na testovací sadě C_2^f

sledky dříve dobrých predikcí. Zdůvodnění můžeme vidět na grafu 7.3 – problémem je malé množství dat s vysokými chybami, které průměr pousouvají z oblasti dobrých predikcí. Vyřešení tohoto problému je z výstupu NN obtížné, neboť špatná predikce může vyjít téměř pro jakoukoli akci nehledě na očekávanou hodnotu *loss* kvůli výskytu outlierů. Potenciální cestou k řešení by mohlo být přidání nové schopnosti NN odhadovat důvěru ve svou predikci hodnoty *loss*, což by nám mohlo dát informace pro správné přeškálování predikcí.

Závěr

Cílem této práce bylo nalézt možná řešení reálného problému automatizace predikce výrobních časů. Dobré výsledky mohou výrazně zlepšit efektivitu a snížit tak výrobní náklady. V práci jsme zanalyzovali různé varianty, jak k tomuto problému přistupovat a jaké jsou jejich výhody a nevýhody. Prodiskutovali jsme také různé účely použití založené na reálných typech situací, pro jednotlivé účely jsme následně zkoumali jejich nejvhodnější řešení.

Nejprve jsme se zabývali problémem výrazného šumu a chyb v reálných datech, jež měly pro většinu modelů negativní vliv. Ukázalo se, že dobrým řešením je použít metodu výpočtu Z-skóre pro jejich detekci kvůli kombinaci nízkých výpočetních nároků a pro trénování modelů vhodněji detekované outliery. Alternativním řešením je využití prediktivní metody s nenulovou robustností, která se s malým množstvím problematických dat dovede vypořádat.

První sadou diskutovaných modelů se staly varianty standardní lineární regrese, jejichž výhodou je především vysoká rychlost. Ukázali jsme, že hlavně pro častěji prováděné akce lze dospět k dobré predikci, zejména pak naší kombinovanou verzí kvůli lepšímu pokrytí situací vyskytujících se v datech.

Vzhledem k neschopnosti lineárních regresí nalézt provázanost jednotlivých operací, důsledkem čehož byly špatné predikce pro nové operace, jsme se v další části pokusili tento problém vyřešit využitím clusterizačních technik založených na dostupných informacích o prováděných operacích. Ukázalo se však, že vyjití jen z těchto informací nevede k dobrému rozdělení operací, což bylo způsobeno zejména velmi podobnými popisy operací se zcela odlišnými výrobními časy. Takové operace se lišily různými klíčovými prvky, jejich automatické rozpoznání však bez hlubšího pochopení problematiky modelem neprobíhala.

Nakonec jsme problém provázání řešili s pomocí hlubokých neuronových sítí. Důležitým bodem zde byla diskuze technik pro zpracování různorodých typů dostupných informací nezbytných k dobré predikci. Dle výsledků se tato metoda ukázala jako velmi úspěšná pro nové typy operací, navíc se schopností odhadnout úroveň své chyby.

Navazující výzkum

Zejména u modelů založených na neuronových sítích je řada neprozkoumaných bodů, které by mohly vést k vylepšení výsledků. Bylo by užitečné dosáhnout dodatečného pokroku v problematice odhadování chyby predikce. Zmiňme také možnost neslučování přípravného a jednicového času operací – to se sice může zdát zbytečné, nicméně různé typy operací mohou obsahově sdílet právě jeden z nich, což by tak mohlo ještě mírně zlepšit dosažené výsledky.

Přestože jsme v práci ukázali, že nemá smysl pokoušet se o clusterizaci jen dle nezávislých proměnných, použití jiných technik schopných dobře využít informace ze závislé proměnné může fungovat – neuronové sítě lze konec konců za takovou techniku považovat. Nalezení alternativního přístupu s přímočařejší interpretací svých rozhodnutí by však mohlo přispět k zjednodušení nasazování automatizace do výroby kvůli možnostem snazší lidské kontroly.

Seznam použité literatury

Bai, X. (2012). Robust Linear Regression. Master's thesis, Kansas state university.

Blei, D., Ng, A. a Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, **3**, 993–1022.

Breiman, L. (1984). *Classification and Regression Trees*. Routledge.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. a Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Cole, R., Salowe, J. S., Steiger, W. L. a Szemerédi, E. (1989). An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, **18** (4), 792–810.

Duchi, J., Hazan, E. a Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, **12**, 2121–2159.

Dudley, N. A. (1963). Work-time distributions. *International Journal of Production Research*, **2**, 137–144.

Ester, M., Kriegel, H.-P., Sander, J. a Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, (226–231).

Goodfellow, I., Bengio, Y. a Courville, A. (2016). *Deep Learning*. MIT Press.

Gurevich, P. a Stuke, H. (2019). Pairing an arbitrary regressor with an artificial neural network estimating aleatoric uncertainty. *Neurocomputing*, **350**, 291–306. doi: 10.48550/ARXIV.1707.07287.

Hamel, G. (1905). Eine Basis aller Zahlen und die unstetigen Lösungen der Funktionalgleichung:f(x+y)=f(x)+f(y). *Matematische Annalen*, **60**, 459–462.

Hochreiter, S. a Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Kingma, D. P. a Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, (1–13).

Law, A. M. a Kelton, W. D. (2000). *Simulation Modeling and Analysis*. McGraw-Hill.

Malik, K., Sadawarti, H. a Kalra, G. (2014). Comparative Analysis of Outlier Detection Techniques. *International Journal of Computer Applications*, **97**, 12–21. doi: 10.5120/17026-7318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. a Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

Polgar, K. C. (1996). Simplified time estimation for basic machining operations. Master's thesis, M.I.T.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks: The Official Journal of the International Neural Network Society*, (543–547).

Quinlan, J. R. (2003). Cubist user guide. *Rulequest Research Data Mining Tools*.

Rousseeuw, P. a Leroy, A. (1987). *Robust Regression and Outlier Detection.* Wiley.

Ruben, R. A. a Mahmoodi, F. (2000). Lead time prediction in unbalanced production systems. *International Journal of Production Research*, **38**(7), 1711–1729.

Rumelhart, D. E., Hinton, G. E. a Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Seber, G. A. F. (2003). *Linear Regression Analysis*. John Wiley & Sons Inc.

Stanley, K. O. a Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, **10**(2), 99–127.

Öztürk, A., Kayaligil, S. a Özdemirel, N. E. (2005). Manufacturing lead time estimation using data mining. *European Journal of Operational Research*, **173**(2), 683–700.

Seznam obrázků

Seznam tabulek

A. Přílohy

A.1 Zdrojová data

Reálná vstupní data pocházející z produkční databáze byla anonymizována a zkonvertována do standardních objektů v jazyce Python 3. Jde o množiny $\mathcal{A}, \mathcal{V}$ a $\mathcal{P}$, se strukturou operací a produktů dle tabulek 3.1 a 3.2. Tyto objekty byly serializovány pomocí standardní knihovny `pickle` do souboru `data_final.pickle` přiloženého k práci.

A.2 Zdrojový kód experimentů

Pro provádění všech experimentů včetně tvorby většiny grafů byl využit programovací jazyk Python 3. Vývoj probíhal v prostředí Jupyter notebooku `Data_analysis.ipynb`, který kromě možnosti nezávislého spouštění jednotlivých bloků kódu umožňuje dodatečné grafické značení a uchovávání výstupů jednotlivých bloků včetně náhledů grafů.

A.2.1 Použitý SW a HW

Všechen zdrojový kód byl napsaný v jazyce Python 3, konkrétně verze 3.7.9 v 64bitové variantě. Pro vývoj a experimenty byly využity následující dodatečné knihovny, doinstalované pomocí `pip` verze 20.1.1:

- `jupyter-core` v4.12.0: Backend pro běh vývojového prostředí,
- `notebook` v6.5.2: Frontend, HTML rozhraní pro vývoj v `jupyter`,
- `matplotlib` v3.5.1: Knihovna pro vytváření grafů,
- `numpy` v1.21.5: Matematická knihovna pro efektivní provádění výpočtů,
- `scikit-learn` v1.0.2: Knihovna obsahující implementace mnoha machine-learning modelů a nástroje k trénování modelů splňujících API této knihovny (viz Pedregosa a kol., 2011).
- `tensorflow-gpu` v1.15.0: Knihovna pro práci s tensory s podporou výpočtu na GPU. Použita pro implementaci deep learning řešení z kapitoly 6. Pro správnou funkčnost na Nvidia GPU je třeba doinstalovat odpovídající ovladače (CUDA, CuDNN).
- `tensorboard` v1.15.0: GUI pro analýzu průběhu učení sítí na základě reportování atributů přes API `tensorflow`.

Vývoj byl prováděn především na notebooku s následujícími parametry:

- Procesor Intel Core i7-7820 (4jádrový, multithreading),
- 32 GB RAM typu SODIMM,
- GPU NVIDIA Quadro M2200,

- 1TB SSD disk MX500.

Takto vysoké HW parametry nejsou nezbytné, všechny modely kromě neuronových sítí mají relativně nízké paměťové i časové nároky. Pro neuronové sítě může mít použitý HW vliv na rychlost učení, parametry sítí použité pro provedení finálních experimentů by však měly být dostatečně nízké pro běh i na 1 GB GPU, případně lze použít variantu knihovny `tensorflow` bez podpory GPU za cenu zpomalení výpočtu.

A.2.2 Struktura kódu

Soubor je členěn do jednotlivých bloků, kde každý blok může být jiného typu. Byly využívány typy `code` pro zdrojový kód a `markdown` pro označování funkce jednotlivých částí.

Jednotlivé logické sekce jsou vždy odděleny označovacími buňkami:

- Importy knihoven a dat: Stará se o načtení všech kódem používaných knihoven a o správnou deserializaci dat z přílohy A.1 včetně definice využívaných datových struktur a konstant.
- Analýza vstupních dat: Výpočty pro generování statistických tabulek o vstupních datech.
- Příprava struktur a metrik: Implementace wrapperů pro jednodušší práci s knihovnou `sklearn` a definice funkce pro výpočet *loss*.
- Konstrukce testovacích sad: Jsou zde vytvořeny generátory pro data splňujících požadavky na jednotlivé testovací sady C_i. Zároveň jsou data rozdělena přímo na trénovací a testovací množiny. V této sekci jsou také definovány filtrační metody, tedy Z-skóre a DBSCAN dle kapitoly 3.
- LR algoritmy: Implementace všech algoritmů z kapitoly 4 – část algoritmů převzata z knihoven, část byla naimplementována přímo.
- LDA: Implementace LDA algoritmu z kapitoly 4 a analýza jeho průběhu s vygenerováním grafu 5.1 na základě uložených hodnot entropií při každé iteraci.
- Implementace NN: Kompletní implementace navrhnuté neuronové sítě z kapitoly 6, využívající knihovnu `tensorflow`. Podporuje celé v práci specifikované API – řada možných nastavení embeddingů jednotlivých atributů nebyla při optimalizaci parametrů využita, jejich využití pro další testy je však připravené.
- Experimenty: Ne vždy zcela zarovnaný kód obsahující evaluační metody a spouštějící vyhodnocení modelů ze všech předchozích bloků. Hlavním cílem kódu je vygenerování všech výsledků pro kapitolu 7. Pro jednodušší modely jsou zde definovány metody sloužící k automatickému vyhodnocení úspěšnosti jednotlivých modelů, pro složité modely (zejména NN) jsou některé specifické analýzy výsledků provedeny samostatným kódem v závislosti na dostupných datech.

A.2.3 Dodatečné parametry NN

Jak bylo zmíněno v sekci 7.4, API implementace NN podporuje řadu dalších parametrů nastavitelných při učení. Zmiňme zde dodatečně všechny důležité pro správné chování NN, které ještě nebyly specifikovány:

- `accuracy_bounding_box`: Oříznutí hranic povolené hodnoty $L(a)$ ze sekce 6.4 na $(ABB, 1 - ABB)$. Tato funkcionalita byla přidána pro zabránění divergence pro nějaký neočekávaný typ operace zejména v prvních epochách učení, která mohla způsobit selhání sítě nastavením hodnot v ní na NaN.

- `concatenate`: Zda mají být embeddingy řetězců slepeny za sebe místo sečtení. Nakonec nebylo při hledání optima použito.

- `use_gradient_clipping`, `gradient_clipping_limit`: Bezpečnostní prvek bránící divergenci v případě neočekávaně silného outlieru. Nebylo třeba používat po doplnění bezpečnostních hranic do kritických částí sítě.

- `group_merge_size`: Ekvivalent MA z kapitoly 7 sloužící ke spojení vzácných kategorií. Každý vstupní atribut je sítí zpracováván zvlášť, vzhledem k dodatečným schopnostem není vhodné tento parametr použít jen na typy operací.

- `ignored_product_names`, `ignored_material_names`: Seznamy atributů od produktů a jejich materiálů ignorovaných při učení sítě. Jména materiálů nakonec kvůli HW nárokům nebyla použita. Pomocí modifikace těchto parametrů byly vygenerovány výsledky sítí bez využívání materiálů.

- `model_save_path`: Cesta pro uložení natrénovaného modelu (bez koncovky), viz A.4. Znaky `%` jsou automaticky nahrazovány aktuálním časem k zajištění jednoznačnosti jednotlivých pokusů.

- `summaries_path`: Cesta pro ukládání průběžných statistik pro zobrazování pomocí knihovny `tensorboard`. Shodné chování vůči `%`.

- `load_path`: Při nenulové hodnotě se tato cesta použije k obnovení stavu sítě. Více viz A.4.

- `seed`: Inicializace náhodného generátoru pro síť. Zejména při více vláknech neřeší zcela problém nedeterministického pořadí dotazů z jednotlivých vláken.

- `threads`: Maximální počet vláken použitelných pro výpočet v síti. Míra vlivu může být výrazně ovlivněna aktuální HW specifikací.

- `use_err_prediction`: Přepíná provádění predikce chyby odhadu *loss*. Použito pro experimenty bez této funkcionality.

- `use_initial_lr_raise, initial_lr, initial_lr_raise_time`: Krátké snížení learning rate na začátku trénování pro snížení velikosti skoků vah kvůli špatným predikcím způsobeným náhodnou inicializací sítě.

A.3 Výsledky modelů založených na LR s využitím grid search

V příloze ve složce `lr_models` lze nalézt parametry a výsledky optimálních modelů jednotlivých verzí LR nalezených pomocí grid search API `sklearn` uložené v `pickle` formátu. Tyto uložené modely lze ve zdrojovém kódu v sekci experimentů načíst a použít přiložené funkce k jejich zobrazení.

V této složce se vyskytují soubory pro každou kombinaci pro kterou byl proveden grid search. Tabulky 7.6, 7.7, 7.8, 7.9 a 7.10 zobrazují všechny jejich výsledky. Jmenná konvence souborů dodržuje formát
`<typ_regresoru>_<testovaci_sada>_<filtracni_mod>.pickle`.

K zobrazení výsledků je lze předat funkci `run_regressor_search` v sekci experimentů jako uložené výsledky, čímž se přeskočí provádění automatického grid search a namísto toho se využijí uložená data.

A.4 Uložené modely sítí

V příloze ve složce `nn_models` lze nalézt natrénované modely jednotlivých typů zkoumaných neuronových sítí z tabulky 7.14. Soubory dodržují jmennou konvenci
`nn_<typ_modelu>_<testovaci_sada>_<cas_spusteni>.<koncovka>`, kde soubory s koncovkami `data-00000-of-00001`, `index` a `meta` obsahují natrénované hodnoty sítě a soubor `pickle` všechny její nezbytné proměnné a parametry, aby byla zajištěna možnost deterministicky obnovit modely použité k experimentům.

Pro načtení modelu je třeba funkci `fit` třídy `Network` předat jako parametr `load_path` cestu k těmto souborům bez přípony. Je-li tento parametr specifikován, jsou všechny ostatní parametry a datový vstup ignorovány a model je nastaven do stavu shodného při provádění jeho uložení, tj. očekává zavolání funkce `predict` pro provedení predikce.

Poznámka. Přestože je garantováno deterministické chování při načtení uloženého NN modelu, nepodařilo se garantovat zcela shodný průběh dvou spuštění funkce `fit` bez něj i při použití shodných parametrů včetně hodnoty seedu. Rozdíly jsou nejspíše způsobeny mnohovláknovým zpracováním učení na GPU, kdy může docházet k nekonzistentnímu pořadí zisku hodnot z náhodného generátoru jednotlivými vlákny. Dle pozorování však případné rozdíly dosahovaly změn v hodnotách *loss* v řádu 10^{-4}, neměla by to tak být překážka pro nezávislé ověření výsledků včetně samotného procesu učení.