VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA LOGOVACÍCH SOUBORŮ A DETEKCE ANOMÁLIÍ

LOG FILE ANALYSIS AND ANOMALY DETECTION

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE **Bc. HUNG DO**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. PETR MATOUŠEK, Ph.D., MA**
SUPERVISOR

BRNO 2024

Abstrakt

V dnešní době mnoho rozsáhlých aplikací běží na distribuovaných systémech jejich správa je čím dál víc obtížnější. Administrátoři většinou ručně procházeli logovací záznamy, aby získali přehled o tom, co se v systému děje. Tento přístup je bohužel v moderních systémech již nemožné použít z důvodu komplexity systémů a velkému množství záznamů, které tyto systémy dokážou rychle generovat. Proto bylo za potřebí najít způsob, jak systémové anomálie automaticky a spolehlivě detekovat. V této práci se tomuto tématu budeme věnovat, rozebereme, jakým způsobem logovací záznamy předzpracovat do strukturované podoby, co je potřeba k tomu, aby se data dala předat ke strojovému učení. V druhé části práce pak popisuje implementaci nástroje `log-monitor` a testování jeho efektivnosti.

Abstract

Many large applications run on distributed systems these days and are becoming more and more troublesome to manage. Traditionally, admins have to manually check all the log contents to learn what is happening in these systems. This approach is not applicable in modern systems due to the complexity of these systems and their large amount of generated log records. Thus it is needed to create a tool that can detect anomalies automatically and reliably. In this paper, we will discuss this topic and analyze how these tools parse log records and what is required to convert these data into the format accepted by machine learning algorithms. In the second half of the paper, we will discuss the implementation and evaluation of `log-monitor` tool.

Klíčová slova

OpenStack, logovací soubory, předzpracování, detekce anomálií, učení bez učitele, LogHub

Keywords

OpenStack, log files, preprocessing, anomaly detection, unsupervised learning, LogHub

Citace

DO, Hung. *Analýza logovacích souborů a detekce anomálií.* Brno, 2024. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Petr Matoušek, Ph.D., MA

Analýza logovacích souborů a detekce anomálií

Prohlášení

Prohlašuji, že jsem tento projekt vypracoval samostatně. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Hung Do
21. dubna 2024

Poděkování

Chtěl bych poděkovat náhodnému Irovi na internetu, který mi ukázal, jak zprovoznit Isolation Forest algoritmus. Jeho desetiminutové video na YouTube bylo velkým přínosem do tohoto projektu. Též bych chtěl poděkovat kuchařkám ve školní menze. Bez všech těch kuřecích plátků s hranolkami, které jsem za poslední týden snědl, bych tento projekt nedokázal dokončit.

Obsah

Kapitola 1

Úvod

V dnešní době je pro administrátory více a více náročné spravovat velké distribuované systémy. Každý systém generuje nesmírné množství logovacích záznamů a tudíž je nemožné, aby jeden člověk ručně projížděl tyto záznamy a hledal v nich nějaké nesrovnalosti. Pro správný a bezpečný chod těchto systémů je důležité mít nástroj, který by dokázal spolehlivě detekovat anomálie v reálném času. Každá pozdě odchycená chyba by mohla vést k velkým finančním ztrátám pro firmy.

Mnoho vědců po celém světě tudíž začla zkoumat, jak by bylo možné tyto anomálie v logovacích záznamech detekovat. V běžných nedistribuovaných systémech administrátoři manuálně procházeli logovací soubory systému a hledali v nich podivné záznamy (např. podle závažnosti záznamu nebo jiných kritérií). Tato práce vyžadovala i expertní znalost systému. Bohužel tato metoda přestává být v dnešní době být dostatečná, pokud bychom provozovali distribuované systémy, a to proto, že:

- tyto systémy jsou velmi komplexní a je nemožné pro jednoho člověka, aby pochopil vše, co se v systému děje,
- rychlost, jakými si záznamy vytvářejí, je vysoká (mnohdy se generuje pro jeden jediný systém generuje několik desítek záznamů za sekundu),
- jednotlivé záznamy nemají jednotný formát (nejsou strukturované) a není tedy možné vytvořit jednoduché nástroje na automatizaci této práce.

Mnoho vědeckých článků se zabývají tímto tématem. V roce 2016 vydal výzkumník Shilin He, a jeho tým, práci s názvem *Experience Report: System Log Analysis for Anomaly Detection* [2], kde popisují způsob zpracování logovacích záznamů a použití umělé inteligence, která se snaží detekovat anomálie ve vstupních datasetech. Celý proces se dá rozdělit na čtyři podúkoly:

1. sběr logovacích záznamů (log collection),
2. předzpracování těchto záznamů (log parsing),
3. tvorba matic příznaků za pomocí událostí (feature extraction)
4. a trénování modelu a detekce anomálií (anomaly detection).

Jelikož jsou logovací záznamy nestrukturované, je potřeba je nejprve **předzpracovat**. Snažíme se ze záznamů dostat jak všeobecná informace (např. čas záznamu, typ závažnosti

záznamu, nebo ID procesu apod.), tak i rozpoznat, o jaký druh události se jedná. Pokud bychom měli šablony všech druhů událostí, můžeme všem záznamům přiřadit ID události. Bohužel ne vždy je tomu tak a proto existuje mnoho parsovacích algoritmů, které se snaží ze záznamů tyto šablony extrahovat. Mezi příklady algoritmů patří SLCT, IPLoM, LKE, LogSig [5], Log Mining [1] a jiné. Porovnáním parsovacích algoritmů se zabývá práce *Tools and Benchmarks for Automated Log Parsing* od Jieming Zhu a jeho týmu [7].

V další části je zapotřebí **získat matici příznaků**. Jedná se o matici skládající se z číselných údajů, které se předají trénovacím algoritmům ke strojovému učení. Každý řádek matice představuje pozorované okno (nějaké časové období), ve kterém se sčítají výskyty jednotlivých druhů událostí. Existují 3 metody, podle které se tyto matice vytvářejí. Metoda s *fixní velikosti okna* sčítá výskyty jednotlivých událostí během fixní časové délky. Pokud tedy chce uživatel pozorovat záznamy po dobu 5 minut, pak každých minut se vygeneruje řádek matice s počty výskytu jednotlivých událostí v daném časovém slotu. Další metodou je metoda s *posuvným časovým oknem*. Tato metoda má kromě parametru velikosti okna i druhý parametr, který definuje, za jak dlouho se má spustit další pozorování (tudíž může v jednu chvíli probíhat více pozorování zároveň). Pokud budeme například chtít pozorovat každých 10 minut okno velikosti jedné hodiny, budou se generovat řádky v časovém rozmezí `00:00-01:00`, `00:10-01:10`, `00:20-01:20` atd. Poslední metoda k sobě shlukuje záznamy, které by mohly patřit do jednoho *sezení* (session). Každé sezení má svůj identifikátor, který nástroj musí extrahovat ze záznamů. Časové okno sezení je pak doba mezi prvním a posledním výskytem identifikátoru v záznamech (začátek a konec komunikace/instance).

V poslední části dochází ke **strojovému učení k detekci anomálií**. Pokud máme k dispozici anotovaná data, je možné použít metody učení s učitelem, mezi které patří například logistická regrese, rozhodovací stromy (decision trees), nebo i jiné metody. V případě neanotovaných dat můžeme shlukování (clustering), PCA (Principal Component Analysis), nebo Invariant Mining [4], který hledá vztahy mezi záznamy a sestavuje „flow chart“ (vývojový graf).

Tato práce popisuje implementaci a testování takového nástroje za pomocí Python skriptu. Podle metriky F-measure se pak výsledky srovnají s ostatními implementacemi.

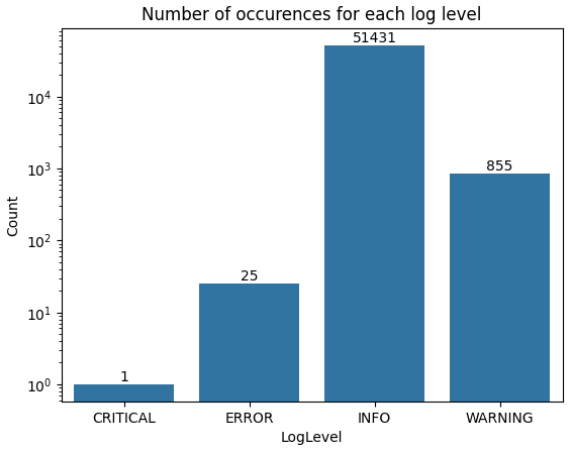
Kapitola 2

Analýza vybraného datasetu

Pro tento projekt jsem vybral OpenStack[1] dataset z veřejně dostupného repozitáře LogHub [6]. Archiv s datasety obsahuje celkem čtyři soubory:

- `anomaly_labels.txt` Obsahuje seznam instancí v `openstack_abnormal.log`, které byly označeny za anomálie.
- `openstack_normal1.log` - Logovací soubor, který obsahuje přes **52 tisíc** záznamů, které byly nasbírány během **6,5 hodiny**.
- `openstack_normal2.log` - Logovací soubor, který obsahuje přes **137 tisíc** záznamů, které byly nasbírány během **21 hodin**.
- `openstack_abnormal.log` - Logovací soubor, který obsahuje přes **18 tisíc** záznamů, které byly nasbírány během **2,5 hodiny**.

Podle souboru `anomaly_labels.txt` byly během běhu programu manuálně přidány 4 anomálie. Naším úkolem bude vytrénovat model, který tyto instance dokáže detekovat.

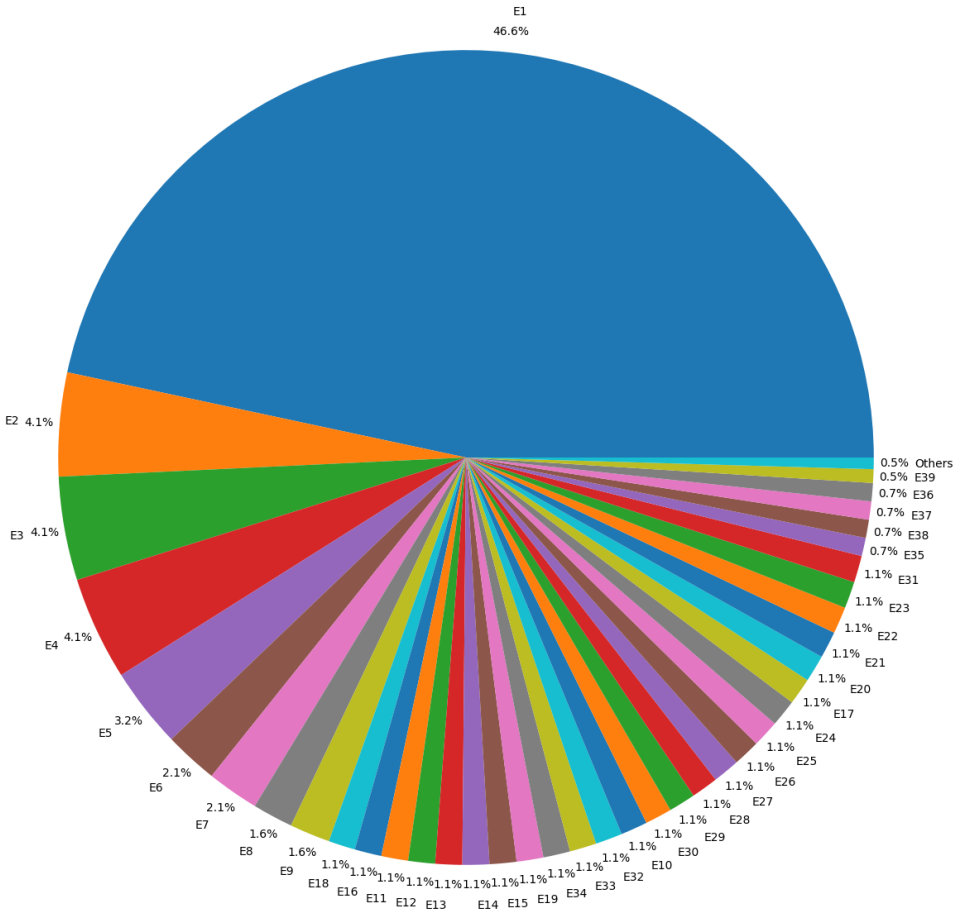


Obrázek 2.1: Počet záznamů podle závažnosti záznamů.

Pro analýzu samotných záznamů jsem použil data ze souboru `openstack_normal.log`. Každý záznam se skládá ze 7 částí: druh záznamu, čas záznamu, ID procesu, důležitost

[1] https://www.openstack.org/

záznamu, komponenta, adresa a obsah záznamu (zpráva). Z oficiálních stránek LogHubu jsem si stáhl šablonu[2] všech možných logovacích zpráv, které se můžou objevit v daném datasetu. Nicméně vybraný dataset obsahoval i zprávy, které šablona na seznamu neměla, proto jsem je musel ručně doplnit. Celkem měl dataset 47 druhů zpráv (40 nejčastějších viz. tabulka 2.1). Jednotlivé druhy událostí jsem seřadil podle početnosti v datasetu pro rychlejší zpracování mým nástrojem.



Obrázek 2.2: Poměr jednotlivých druhů zpráv/událostí.

Při analýze vybraného logovací souboru jsem zjistil, že obsahuje primárně obsahuje záznamy typu *INFO* (viz. graf 2.1) a události typu *E1* (viz. graf 2.2). Můžeme si všimnout, že se všechny události *E10* až *E34* vyskytují ve stejném poměru. To nám může naznačit, že jednotlivé komunikace mají jasně dané pořadí zpráv. Po podrobném prozkoumání logovacích záznamů jsem objevil, že jednotlivé komunikace začínají událostí *E18* a končí *E10*. Pro extrakci příznaků tedy použijeme metodu **session window** (okno událostí podle sezení).

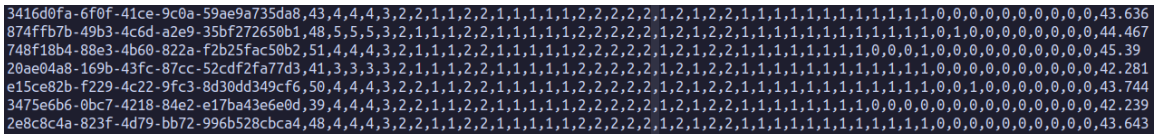
Proto navrhuji, aby se matice příznaků vytvářely tímto způsobem:

[2] https://github.com/logpai/loghub/blob/master/OpenStack/OpenStack_2k.log_templates.csv

1. Ze záznamů si vytáhneme ID sezení. Tato sezení pro nás budou představovat řádky v matici.

2. Každý řádek matice bude obsahovat *ID sezení*, *počet výskytů* jednotlivých typů událostí a *doba trvání* komunikace (rozdíl mezi časem začátku a konce komunikace).

Vytvořenou matici pak použijeme ke trénování našeho modelu. Příklad matice událostí nalezneme v obrázku 2.3.



Obrázek 2.3: Příklad matice událostí

EventID	Event Pattern
E1	<*> "GET <*>"status: <*> len: <*> time: <*>.<*>
E2	image <*> at (<*>): checking
E3	image <*> at (<*>): in use: on this node <*> local, <*> on other nodes sharing this instance storage
E4	Active base files: <*>
E5	<*> "POST <*>"status: <*> len: <*> time: <*>.<*>
E6	[instance: <*>] VM Resumed (Lifecycle Event)
E7	During sync_power_state the instance has a pending task (<*>). Skip.
E8	Removable base files: <*>
E9	Unknown base file: <*>
E10	[instance: <*>] VM Stopped (Lifecycle Event)
E11	[instance: <*>] Creating image
E12	[instance: <*>] VM Paused (Lifecycle Event)
E13	[instance: <*>] Took <*>.<*> seconds to build instance.
E14	[instance: <*>] Instance spawned successfully.
E15	[instance: <*>] VM Started (Lifecycle Event)
E16	[instance: <*>] Took <*>.<*> seconds to spawn the instance on the hypervisor.
E17	[instance: <*>] Claim successful
E18	[instance: <*>] Attempting claim: memory <*> MB, disk <*> GB, vcpus <*> CPU
E19	[instance: <*>] disk limit not specified, defaulting to unlimited
E20	[instance: <*>] memory limit: <*>.<*> MB, free: <*>.<*> MB
E21	[instance: <*>] Total disk: <*> GB, used: <*>.<*> GB
E22	[instance: <*>] Took <*>.<*> seconds to deallocate network for instance.
E23	[instance: <*>] vcpu limit not specified, defaulting to unlimited
E24	Creating event network-vif-plugged:<*>-<*>-<*>-<*>-<*> for instance <*>
E25	[instance: <*>] Total vcpu: <*> VCPU, used: <*>.<*> VCPU
E26	[instance: <*>] Total memory: <*> MB, used: <*>.<*> MB
E27	[instance: <*>] Terminating instance
E28	<*> "DELETE <*>"status: <*> len: <*> time: <*>.<*>
E29	[instance: <*>] Took <*>.<*> seconds to destroy the instance on the hypervisor.
E30	[instance: <*>] Instance destroyed successfully.
E31	Removing base or swap file: <*>
E32	[instance: <*>] Deletion of <*> complete
E33	HTTP exception thrown: No instances found for any event
E34	[instance: <*>] Deleting instance files <*>
E35	Final resource view: name=<*> phys_ram=<*> used_ram=<*> phys_disk=<*> used_disk=<*> total_vcpus=<*> used_vcpus=<*> pci_stats=[]
E36	Total usable vcpus: <*>, total allocated vcpus: <*>
E37	Compute_service record updated for <*>
E38	Auditing locally available compute resources for node <*>
E39	Base or swap file too young to remove: <*>
E40	Successfully synced instances from host '<*>'.

Tabulka 2.1: 40 nejčastějších typů záznamů, které se objevují v logovacích souborech.

Kapitola 3

Implementace

Program používá knihovny **Pandas**[1] pro práci s velkými datasety a **scikit-learn**[2] pro trénování modelu. Mezi dalšími použitými knihovnami patří *argparse* pro zpracování programových argumentů, *NumPy* jako pomocná knihovna k *Pandas*, nebo *re* jako knihovna pro práci s regulárními výrazy.

Implementace programu `log-monitor` by se dala rozdělit na tři části. V první části dochází k načtení a extrakce dat z logovacích souborů. Program obsahuje seznam regulárních výrazů jménem `EVENT_REGEX`, ve kterém jednotlivé výrazy reprezentují jednotlivé události v tabulce 2.1. Funkce `parse_line` pak se pak snaží k jednotlivým řádkům souborů přiřadit správný ID události. Funkce též parsuje zbytek informací z řádku (viz. struktura záznamu v kapitole 2). Poté, co se všechny řádky převedou strukturované kolekce, jsou kolekce uloženy do `pandas.DataFrame` objektu. Výstup této části je k nalezení v souborech `training_log_structure.csv` a `testing_log_structure.csv`.

V druhé části dochází k tvorbě matice příznaků. Z daného *DataFrame* objektu si získáme seznam všech ID instancí. U každé instance zjistíme řádky začátku komunikace (událost typu *E18*) a konce komunikace (událost typu *E10*). Vezmeme všechny řádky, které se nachází mezi těmito řádky a spočítáme matici událostí (pro každý typ události spočítáme počet jejich výskytu v daném seznamu řádků). Nakonec spočítáme délku celé komunikace v sekundách. Výsledná matice příznaků se pak skládá ze všech instancí v logovacím souboru. Samozřejmě pokud nějaké instance chybí začátek nebo konec komunikace (často na začátku nebo na konci logovacího souboru), pak je instance vynechána z matice příznaků. Výstup této části je k nalezení v souborech `training_event_matrix.csv` a `testing_event_matrix.csv`.

V poslední části dochází k samotnému trénování a vyhodnocení modelu na testovacích datech. K trénování modelu byla použita metoda **Isolation Forest**, která slouží k detekci odlehlých bodů. Budeme očekávat, že výpočet odlehlých bodů budou nejvíce záviset na sloupcích *E42* (Error úroveň závažnosti), *E46* (Critical úroveň závažností), *délka komunikace* a speciální sloupec *E-1*, který reprezentuje událost, který program nezná a nedokázal ho zpracovat při extrakci dat (v první části). Nakonec na tento model zavoláme funkci `.fit()`, kterému předáme náš trénovací dataset a necháme ho se učit. Výsledný model nám pak povoluje zavolat dvě funkce pro zjištění odlehlých bodů (v našem případě instancí). Funkce `decision_function` je klasifikační metoda a vrací hodnotu mezi -1 a 1. Nižší hodnota znamená, že se bod (instance) více odlišuje od většiny bodů (instancí) v datasetu

[1] https://pandas.pydata.org/
[2] https://scikit-learn.org/stable/

(neboli tím víc si je model jistý, že se jedná o odlehlý bod/anomálii). Funcke `predict` je pak signum této funkce (výstupem funkce signum je -1 pokud model hodnotí bod jako anomálii a 1 jako normální bod). Trénovací algoritmus nalezneme ve výpise 3.1.

```
ANOMALY_INPUTS = ["E42", "E46", "TimeDiff", "E-1"]

def train_model(
    train_df: pd.DataFrame, contamination: float, random_state
) -> IsolationForest:
    model_IF = IsolationForest(
        contamination=contamination,
        random_state=random_state
    )
    model_IF.fit(train_df[ANOMALY_INPUTS])
    train_df["AnomalyScore"] = \
        model_IF.decision_function(train_df[ANOMALY_INPUTS])
    train_df["Anomaly"] = model_IF.predict(train_df[ANOMALY_INPUTS])
    return model_IF
}
```

Výpis 3.1: Algoritmus trénování modelu a detekce anomálií v datasetu.

Kapitola 4

Testování

Testování probíhalo na datasetu `openstack_abnormal.log`. V něm se podle souboru `anomaly_labels.txt` nachází čtyři anomální instance virtuálních strojů, které autoři datasetu manuálně vložili. Program `log-monitor` jsem spouštěl s těmito parametry:

- `-training ./OpenStack/openstack_normal2.log` - vstupní trénovací soubor,
- `-testing ./OpenStack/openstack_abnormal.log` - vstupní testovací soubor,
- `-c 0.05` - procento zastoupení anomálií v trénovacím souboru,
- `-rs 42` - magická konstanta sloužící k reprodukci výsledků,
- `-th 0` - program vypíše všechny instance, u kterých je anomální skóre nižší než 0,
- `--export-log-structure` - program vygeneruje CSV soubor s zpracovanými daty po extrakci dat logovacích souborů,
- `--export-event-matrix` - program vygeneruje CSV soubor s maticemi událostí,
- `--export-anomaly` - program vygeneruje CSV soubor s předpovědi modelu (zda je instance anomálie, nebo není).

Program běžel celkem 31 sekund a detekoval celkem 14 anomálií (viz. výpis v 4.2). Čím je skóre anomálie nižší, tím si je model víc jistý, že je instance anomálie. Výstup je seřazen vzestupně podle anomálního skóre (největší anomálie jsou na začátku). Výsledek můžeme ještě více vyfiltrovat pomocí `-th -0.1` možnosti, které nám ukáže jen prvních 5 instancí.

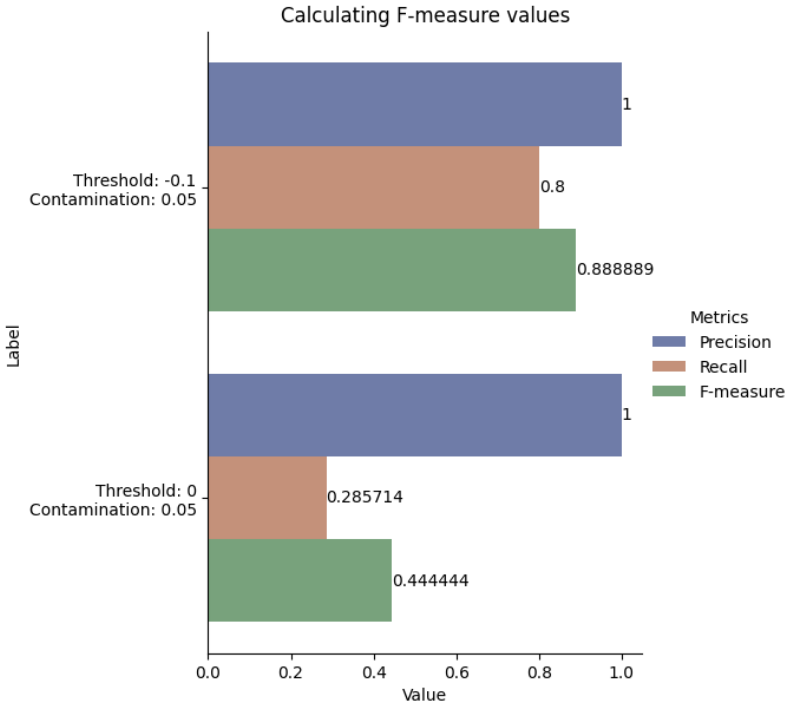
```
The following VM instances have injected anomalies as observed
in openstack_abnormal.log.

544fd51c-4edc-4780-baae-ba1d80a0acfc
ae651dff-c7ad-43d6-ac96-bbcd820ccca8
a445709b-6ad0-40ec-8860-bec60b6ca0c2
1643649d-2f42-4303-bfcd-7798baec19f9
```

Výpis 4.1: Seznam anomálních instancí.

```
                                 InstanceID  AnomalyScore
544fd51c-4edc-4780-baae-ba1d80a0acfc     -0.164556
ae651dff-c7ad-43d6-ac96-bbcd820ccca8     -0.164556
a445709b-6ad0-40ec-8860-bec60b6ca0c2     -0.164556
d7d68893-c44f-4e7c-8d4b-0e7801d62776     -0.164556
1643649d-2f42-4303-bfcd-7798baec19f9     -0.164556
9c9c9fd8-bef9-4fb4-835f-6be8ba1d20b9     -0.046350
21354a9e-a5b1-4412-a444-27374721659b     -0.039912
874ffb7b-49b3-4c6d-a2e9-35bf272650b1     -0.036276
58da1c37-c65a-4240-8ad8-9223470084f1     -0.029541
25415e12-e0a9-4ee1-b5d2-0e6f0165b4f2     -0.029541
5043956c-a09d-4431-8391-452ec5582ee0     -0.014644
53edb4bd-93dd-4237-8c64-ba8d8f5e3b34     -0.011195
7e67a1c0-f296-458e-9e2c-1d96afdf3051     -0.009521
748f18b4-88e3-4b60-822a-f2b25fac50b2     -0.009204
```

Výpis 4.2: Seznam instancí, které model označil za potencionální anomálie.

Můžeme zde vidět, že z 5 náš model dokázal detekovat všechny 4 instance, které autoři datasetu označili za anomálie (viz. výpis v 4.1). Po podrobnějším prozkoumání jsem zjistil, že všechny tyto anomální instance běžely o pár sekund déle, než byla průměrná doba života virtuálního stroje (kolem 40 až 45 sekund).



Obrázek 4.1: Jednotlivé výsledky F-measure hodnot.

Efektivitu našeho modelu ohodnotíme pomocí F-measure[1] metriky. Ta se skládá ze 3 hodnot: *precision* hodnota nám říká, kolik procent z celkového počtu anomálií náš model

[1] https://en.wikipedia.org/wiki/F-score

dokázal detekovat, *recall* hodnota zase popisuje, kolik procent z počtu anomálií, které náš model detekoval, jsou opravdové anomálie, a *F-measure* hodnota, která indikuje harmonický průměr precision a recall.

$$Precision = \frac{\texttt{Počet správně detekovaných anomálií}}{\texttt{Celkový počet opravdových anomálií}}$$

$$Recall = \frac{\texttt{Počet opravdových anomálií z výběru}}{\texttt{Počet detekovaných anomálií}}$$

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

V grafu 4.1 je vidět porovnání výsledků stejného modelu s odlišnou prahovou hodnotou. Pokud bychom za anomálie brali všechny instance, které nám model tak klasifikoval (prahová hodnota je 0), pak bychom dostali mnoho false-positive výsledků a efektivita by klesla na 44.44%. Nicméně pokud by nás zajímali jenom instance se skóre anomálie nižší než $-0,1$, pak budou detekce mnohem přesnější a dostaneme efektivitu model na 88.88%. Tento výsledek je porovnatelný s výsledky práce od pana Kalaki, a jeho týmu, z Shahid Beheshti University v Tehránu [3]. Nutno zde ale podotknout, že jeho tým použil prezentoval výsledky na jiném datasetu[2], který můj skript nedokáže zpracovat z důvodu jiného formátu záznamů (v záznamech se nenachází počáteční signatura záznamu, kterou jsem rozebíral v kapitole 2).

[2] https://github.com/ParisaKalaki/OpenStack-logs

Kapitola 5

Závěr

Mým úkolem bylo vytvořit program, který natrénuje model k detekci anomálií. Program nejprve načte logovací soubory a vytvoří matice událostí za pomoci Session ID jednotlivých instancí (ID sezení). Pomocí matice se pak metodou učení bez učitele (Isolation Forest) vytrénuje model, který nám bude jednotlivá sezení klasifikovat jako normální nebo abnormální (anomálie).

Z výsledného testování bylo zjištěno, že model vskutku dokáže tyto anomálie částečně odhalit s určitou přesností. Přesnost detekcí jsem spočítal pomocí F-skóre (F-measure) metriky z testovacích dat nám v nejlepším případě vyšlo 88.8%. Nicméně tyto výsledky se nedají interpretovat tak jednoduše, jak by se jen zdálo. Uživatel si musí předem nastavit prahovou hodnotu skóre anomálie, která ho bude zajímat (v našem případě nás zajímaly jenom instance, u kterých bylo skóre anomálie nižší než $-0,1$). Jelikož algoritmus Isolation Forest slouží jenom k detekci odlehlých bodů, můžeme očekávat, že program bude klasifikovat mnoho false-positive instancí (instance, které model klasifikoval za anomálii, nejsou anomáliemi). Důležité je též připomenout fakt, že dataset, který byl použit pro tento projekt, byl nejspíš nedostatečně malý a pokrýval jenom zlomek možných situací, které by mohly nastat.

Pokud bych měl výsledky shrnout, tak bych řekl, že i když model dobře funguje nad oficiálně dostupných datasetech, nemůžu ale zaručit, že je vybraná metoda dostatečně přesná na to, aby ji bylo možné použít v reálném provozu.

Literatura

[1] He, P., Zhu, J., He, S., Li, J. a Lyu, M. R. An Evaluation Study on Log Parsing and Its Use in Log Mining. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016, s. 654–661. DOI: 10.1109/DSN.2016.66.

[2] He, S., Zhu, J., He, P. a Lyu, M. R. Experience Report: System Log Analysis for Anomaly Detection. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, s. 207–218. DOI: 10.1109/ISSRE.2016.21.

[3] Kalaki, P. S., Shameli Sendi, A. a Abbasi, B. K. E. Anomaly detection on OpenStack logs based on an improved robust principal component analysis model and its projection onto column space. *Software: Practice and Experience*. 2023, sv. 53, č. 3, s. 665–681. DOI: https://doi.org/10.1002/spe.3164. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3164.

[4] Lou, J.-G., Fu, Q., Yang, S., Xu, Y. a Li, J. Mining invariants from console logs for system problem detection. Leden 2010.

[5] Tang, L., Li, T. a Perng, C.-S. LogSig: generating system events from raw textual logs. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 2011, s. 785–794. CIKM '11. DOI: 10.1145/2063576.2063690. ISBN 9781450307178. Dostupné z: https://doi.org/10.1145/2063576.2063690.

[6] Zhu, J., He, S., He, P., Liu, J. a Lyu, M. R. *Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics*. 2023.

[7] Zhu, J., He, S., Liu, J., He, P., Xie, Q. et al. *Tools and Benchmarks for Automated Log Parsing*. 2019.