SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta elektrotechniky a informatiky

Evidenčné číslo: FEI-16607-111186

Vysvetliteľná klasifikácia kategórií malvéru

Diplomová práca

2025 **Bc. Viktor Bojda**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta elektrotechniky a informatiky

Evidenčné číslo: FEI-16607-111186

Vysvetliteľná klasifikácia kategórií malvéru

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Školiteľ:	Ing. Peter Švec, PhD.

2025 **Bc. Viktor Bojda**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025
Evidenčné číslo: FEI-16607-111186

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Viktor Bojda**
ID študenta:	111186
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Peter Švec, PhD.
Vedúci pracoviska:	doc. Ing. Milan Vojvoda, PhD.
Miesto vypracovania:	Ústav informatiky a matematiky

Názov práce: **Vysvetliteľná klasifikácia kategórií malvéru**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je implementovať klasifikátor, ktorý bude vedieť rozpoznávať konkrétne kategórie malvéru (botnet, ransomware a pod.). Hlavnou náplňou práce bude vytvoriť viaceré modely založené na algoritmoch strojového učenia a aplikovať rôzne vysvetliteľné medódy.

Úlohy:

1. Naštudujte si problematiku vysvetliteľných metód a detekciu malvéru.
2. Implementujte dané riešenie.
3. Porovnajte dosiahnuté výsledky.
4. Výsledné riešenie zdokumentujte.

Zoznam odbornej literatúry:

1. SIKORSKI, Michael; HONIG, Andrew. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.* San Francisco: No Starch Press, 2012. 800 s. ISBN 1-59327-290-1.
2. MOLNAR, Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.* Brétigny-sur-Orge : Amazon, 2022. 317 s. ISBN 979-8-41-146333-0.

Termín odovzdania diplomovej práce:	16. 05. 2025
Dátum schválenia zadania diplomovej práce:	24. 03. 2025
Zadanie diplomovej práce schválil:	prof. Dr. Ing. Miloš Oravec – garant študijného programu

Poďakovanie

Rád by som vyjadril úprimnú vďaku svojmu vedúcemu diplomovej práce, Ing. Petrovi Švecovi, PhD., za jeho trpezlivé vedenie, odborné rady a neustálu podporu počas celej tvorby tejto práce.

Abstrakt

Táto práca sa venuje klasifikácii kategórií malvéru na základe dát z dynamickej analýzy, pričom využíva vysvetľovacie metódy na interpretáciu rozhodnutí modelov strojového učenia. V rámci práce boli implementované tri klasifikačné modely: náhodný les, XGBoost a viacvrstvový perceptrón. Na ich trénovanie bol použitý nami vytvorený dataset, ktorý vznikol kombináciou a predspracovaním dvoch verejne dostupných datasetov. Zdokumentovali sme celý proces implementácie vrátane ladenia hyperparametrov a vykonali sme porovnanie a vyhodnotenie dosiahnutých výsledkov. Pre každý z modelov sme analyzovali niekoľko náhodne vybraných vzoriek pomocou troch vysvetľovacích metód: SHAP, Anchors a kontrafaktuálnych vysvetlení. Na záver sme porovnali a zhodnotili praktickú použiteľnosť týchto vysvetľovacích prístupov.

Kľúčové slová

klasifikácia malvéru, strojové učenie, náhodný les, XGBoost, viacvrstvový perceptrón, XAI, vysvetľovacie metódy, SHAP, Anchors, kontrafaktuálne vysvetlenia

Abstract

This thesis focuses on the classification of malware categories based on data from dynamic analysis, utilizing explainability methods to interpret the decisions of machine learning models. Three classification models were implemented as part of the thesis: Random Forest, XGBoost, and Multilayer Perceptron. A custom dataset, created by combining and preprocessing two publicly available datasets, was used for training these models. We documented the entire implementation process, including hyperparameter tuning, and conducted a comparison and evaluation of the obtained results. For each model, we analyzed several randomly selected samples using three explainability methods: SHAP, Anchors, and counterfactual explanations. Finally, we compared and assessed the practical usability of these explainability approaches.

Keywords

malware classification, machine learning, random forest, XGBoost, multilayer perceptron, XAI, explainability methods, SHAP, Anchors, counterfactual explanations

Obsah

Zoznam obrázkov a tabuliek

Zoznam značiek a skratiek

APC	Asynchronous Procedure Call
API	Application Programming Interface
C2	Command & Control
CEGP	Counterfactual Explanations Guided by Prototypes
CSV	Comma-Separated Values
DeepLIFT	Deep Learning Important FeaTures
DLL	Dynamic Link Library
DNS	Domain Name System
DoS	Denial-of-Service
ELF	Executable and Linkable Format
ELU	Exponential Linear Unit
FACE	Feasible and Actionable Counterfactual Explanations
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GPU	Graphics Processing Unit
GSG	Growing Spheres Generation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
KDE	Kernel Density Estimation
LIME	Local Interpretable Model-agnostic Explanations
LORE	Local Rule-based Explainer
MAB	Multi-Armed Bandit
MD5	Message-Digest Algorithm 5
MDA	Mean Decrease in Accuracy
MDI	Mean Decrease in Impurity
MLP	Multilayer Perceptron
MSVC	Microsoft Visual C++
OOB	Out-of-Bag
P2P	Peer-to-Peer
PE	Portable Executable

RAM	Random-Access Memory
ReLU	Rectified Linear Unit
RF	Random Forest
SGD	Stochastic Gradient Descent
SHA	Secure Hash Algorithm
SHAP	Shapley Additive Explanations
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WSL	Windows Subsystem for Linux
XAI	Explainable Artificial Intelligence
XGB	Extreme Gradient Boosting

Úvod

V súčasnom kybernetickom prostredí, ktoré sa vyznačuje rýchlo sa vyvíjajúcimi hrozbami a rastúcou mierou sofistikovanosti útokov, už nestačí len identifikovať, že určitý softvér je škodlivý (malvér). Pre efektívne riešenie bezpečnostných incidentov je čoraz dôležitejšie vedieť, o aký typ malvéru sa jedná. Klasifikácia malvéru do konkrétnych kategórií, ako sú napríklad trojany, červy či spajvér, zohráva kľúčovú úlohu pri identifikácií jeho správania, cieľov a potenciálneho dopadu. Znalosť presnej kategórie tak umožňuje špecializovaným bezpečnostným tímom rýchlejšie diagnostikovať vzniknutý incident, podľa toho ho prioritizovať a navrhnúť konkrétne protiopatrenia, namiesto generických a často neefektívnych zásahov. Využitie modelov strojového učenia pri tejto úlohe je dnes už bežnou praxou, keďže ponúkajú vysokú presnosť a schopnosť pracovať s rozsiahlymi a komplexnými dátami.

S rastúcou zložitosťou týchto modelov však zároveň rastie aj problém ich vysvetliteľnosti. Algoritmy založené na súborových metódach či hlboké neurónové siete síce môžu dosahovať vysokú predikčnú úspešnosť, no fungujú ako tzv. *čierne skrinky*. Nie je teda zrejmé, na základe čoho model prijal konkrétne rozhodnutie. Obzvlášť v oblasti kybernetickej bezpečnosti, kde môžu mať nesprávne rozhodnutia závažné následky, je kritické vedieť, prečo bol malvér zaradený do konkrétnej kategórie. Bez toho totiž hrozí riziko, že modely budú reagovať na základe nerelevantných dát, čím môžu viesť k chybným interpretáciám a tým aj strate dôvery zo strany odborníkov.

Táto práca sa preto zameriava na vysvetliteľnú klasifikáciu kategórií malvéru na základe dát z dynamickej analýzy. V prvej kapitole sa venujeme malvéru a jednotlivým kategóriám, na ktoré sa delí, pričom pre každú z nich si uvedieme stručnú charakteristiku. Prejdeme si tiež proces analýzy malvéru a rozdiel medzi statickou a dynamickou analýzou. V druhej kapitole si potom priblížime algoritmy strojového učenia náhodný les, XGBoost a viacvrstvový perceptrón. Predstavíme si spôsob ich fungovania a hlavné silné a slabé stránky. V tretej kapitole sa oboznámime s vysvetliteľnosťou v strojovom učení. Vysvetlíme si základné pojmy a kľúčové koncepty, ktoré sú dôležité pre jej pochopenie. Predstavíme si vysvetľovacieho algoritmy SHAP, Anchors a Kontrafaktuálne vysvetlenia, kde si vysvetlíme ako fungujú a na čo sú vhodné. V štvrtej kapitole si uvedieme aké požiadavky sme mali na použité dáta a z pomedzi akých datasetov sme si vyberali. Dôkladne si potom prejdeme celý proces predspracovania dát. V piatej kapitole sa už venujeme samotnej implementácii klasifikačných modelov. Najprv si však stručne uvedieme metódy, ktoré boli použité na overovania a hodnotenie výsledkov. Nasledujú časti, v ktorých podrobne popíšeme najprv proces transformácie dát a potom implementáciu modelov náhodný les, XGBoost a viacvrstvový perceptrón. Na konci

kapitoly sme vykonali porovnanie a vyhodnotenie výsledkov modelov. V poslednej, šiestej kapitole sa venujeme aplikácii vysvetľovacích metód na vytvorené modely, kde si pre náhodne vybrané vzorky urobíme komplexnú analýzu výstupov z metód. Kapitolu uzatvárame porovnaním a vyhodnotením vysvetľovacích metód.

1 Malvér

Malvér (angl. *malicious software*, skr. *malware*) je zastrešujúci pojem, pod ktorým všeobecne rozumieme akýkoľvek softvér špeciálne navrhnutý na úmyselné spôsobenie škody pre systém či používateľa [1]. Motivácií na takéto konanie je mnoho. Najčastejšie sa jedná o finančný zisk, napríklad prostredníctvom krádeže súkromných informácií alebo vydierania. Ničím výnimočným však nie sú ani aktivizmus a osobné spory, ale aj politický zisk či dokonca špionáž a iné vážnejšie zločiny.

1.1 Vektory šírenia

Efektívna kybernetická obrana začína dôkladnou znalosťou spôsobov, akými dochádza k útokom. Inak to nie je ani v oblasti ochrany pred malvérom. Malvér sa neustále vyvíja a jeho autori čoraz sofistikovanejšie využívajú na jeho šírenie kombináciu technických zraniteľností a vzorcov ľudského správania. Najbežnejším, no zároveň najefektívnejším vektorom útoku zostáva e-mail [2]. Útočníci často posielajú správy so škodlivými prílohami alebo s odkazmi na nebezpečné webové stránky. Takéto e-maily sa tvária autenticky, zvyčajne napodobňujú dôveryhodné inštitúcie, ako sú banky či známe spoločnosti. Typickým predstaviteľom je phishing. Ide o formu sociálneho inžinierstva, ktorej cieľom je prinútiť príjemcu otvoriť nebezpečný odkaz či prílohu, čo zvyčajne vedie k neúmyselnému stiahnutiu malvéru alebo k odhaleniu citlivých informácií. Využíva na to techniky navrhnuté na vyvolanie falošného pocitu naliehavosti alebo zvedavosti, čím zvyšuje pravdepodobnosť úspešného útoku. Útočníci cielia najmä na menej skúsených používateľov, pre ktorých sú na prvý pohľad takéto e-maily ťažko rozpoznateľné ako podvodné.

Jednou z mnohých alternatív pre distribúciu malvéru je aj tzv. drive-by download [3]. Škodlivý softvér je automaticky stiahnutý už pri návšteve nebezpečnej či infikovanej webovej stránky. Od používateľa sa nevyžaduje, aby akceptoval akúkoľvek výzvu na stiahnutie alebo klikol na odkaz. Následne môže využiť zraniteľnosti vo webovom prehliadači či v operačnom systéme, v dôsledku ktorých dôjde k jeho spusteniu aj bez priameho zásahu používateľa. Proces sa vykonáva na pozadí a používateľ si ho často nie je vedomý až do momentu, kým malvér nezačne pôsobiť škody.

1.2 Kategórie malvéru

Ako bolo spomenuté na začiatku, malvér nepredstavuje jednu konkrétnu hrozbu, ale zahŕňa celú škálu rôznych typov škodlivého softvéru. Hoci hranice medzi jednotlivými kategóriami sú v praxi často rozmazané, na základe spoločných znakov a vzorcov

správania ich možno zaradiť do konkrétnych skupín. Niektoré z nich, spolu s ich charakteristikami, si teraz popíšeme [1]:

Advér (reklamný softvér): Prejavuje sa vo forme vyskakovacích (angl. *popup*) okien, reklamných banerov priamo na pracovnej ploche, prípadne v rámci webového prehliadača [4]. Primárnym cieľom autora je generovať zisk zobrazením alebo kliknutím na reklamy. Tie bývajú častokrát zdrojom ďalšieho škodlivého softvéru. Zvyčajne sa šíri ako súčasť frivéru[1] alebo šervéru[2] a je spustený automaticky po inštalácii sprievodného programu používateľom. Aj napriek tomu je všeobecne považovaný za najmenej škodný typ malvéru. Niektorí tvorcovia advéru dokonca právne napadli poskytovateľov antivírusových programov kvôli jeho blokovaniu, pričom argumentovali ich údajnou legitimitou [5].

Backdoor (zadné vrátka): Umožňuje útočníkovi vzdialený, neautorizovaný a nepretržitý prístup ku systému obete. Zámerom je prekonať štandardné bezpečnostné opatrenia s cieľom získať prístup k citlivým dátam či priamo manipulovať so systémom. Backdoor typicky pracuje na pozadí, aby unikol pozornosti používateľa a tým si zachoval pretrvávajúci prístup. Často sa do systému dostáva prostredníctvom škodlivých príloh, podvodných súborov na stiahnutie alebo je integrovaný do iných typov malvéru.

Spajvér (špionážny softvér): Uskutočňuje tajné sledovanie a zaznamenávanie činnosti používateľa. Zhromažďuje citlivé osobné dáta, vrátane prihlasovacích údajov, hesiel a finančných informácií. Môže tiež vykonávať nenápadné presmerovania na phishingové či inak kompromitované webové stránky. Podobne ako backdoor, spajvér pracuje na pozadí a obvykle sa do systému dostáva ako súčasť iného, zdanlivo legitímneho softvéru.

Vírus: Navrhnutý na replikovanie a šírenie prostredníctvom infikovania súborov či programov v systéme. Hlavným zámerom je zničenie alebo znehodnotenie dát, prípadne narušenie celkovej funkčnosti systému. Na aktiváciu vírusu je potrebná interakcia používateľa, napríklad spustenie infikovaného súboru. Najčastejšie sa šíria prostredníctvom škodlivých webových stránok, P2P sietí na zdieľanie súborov alebo infikovaných e-mailových príloh.

Červ: Podkategória vírusu, na rozdiel od neho, je však autonómny a je schopný automatickej replikácie a šírenia bez potreby akejkoľvek interakcie používateľa. Hlavným cieľom je čo najrýchlejšie rozšírenie do maximálneho počtu systémov prostredníctvom zneužívania sieťových alebo softvérových zraniteľností, prípadne slabých prístupových údajov. Na postihnutom systéme môže spôsobovať spomalenie, zahltenie siete alebo

[1] bezplatný softvér
[2] zdieľaný softvér

DoS útoky. Červy sa tiež často využívajú na budovanie botnetov alebo na distribúciu iných typov malvéru.

Trójsky kôň (Trojan): Softvér, ktorý sa tvári ako legitímny, pričom občas jeho činnosť aj reálne vykonáva, no na pozadí vykonáva škodlivé aktivity. Po spustení môže kradnúť citlivé dáta, vytvoriť backdoor do systému, či spúšťať iný škodlivý kód. Trójsky kôň sa nedokáže automaticky sám replikovať, preto využíva na šírenie sociálne inžinierstvo, aby používateľa presvedčil o svojej dôveryhodnosti. Distribuovaný je často cez infikované prílohy, phishingové e-maily alebo na stránkach ponúkajúce nelegálny alebo pirátsky softvér.

Dropper: Typ trójskeho koňa, používaný na inštaláciu (pustenie, angl. „dropping“) ďalšieho malvéru na cieľový systém. Na vyhnutie detekcii antivírusovým programom je jeho škodlivý obsah spravidla zašifrovaný či inak maskovaný v legitímne vyzerajúcom softvéri alebo skripte. Po spustení rozbalí škodlivý kód a nasadí ho do systému obete. Často sa po vykonaní svojej funkcie dropper zo systému automaticky odstráni alebo zamaskuje svoju činnosť tak, aby zahladil stopy.

Downloader: Fungovaním a účelom podobný dropperu, s tým rozdielom, že ďalší škodlivý softvér neobsahuje priamo, ale po spustení ho stiahne z internetu. Downloader sa pripája na vopred definovanú URL adresu alebo na vzdialený C2 server, ktorý ho riadi. Pomocou tohto spojenia môže útočník dodatočne aktualizovať či inak prispôsobiť nainštalovaný malvér podľa konkrétneho scenára útoku alebo aktuálnych potrieb.

Vyššie opísané kategórie boli vybrané najmä preto, že sa im ďalej v práci ešte budeme venovať. Nemožno však opomenúť aj iné známe typy malvéru, akým je napríklad **ransomvér** (zašifruje používateľské dáta a za ich odšifrovanie požaduje platbu), **keylogger** (zaznamenáva stlačené klávesy na zachytenie citlivých informácií), či **rootkit** (poskytuje útočníkovy root prístup, pričom skrýva svoju prítomnosť) a mnohé iné.

1.3 Analýza malvéru

Analýza malvéru je proces, ktorý sa zaoberá systematickým skúmaním škodlivého softvéru so zámerom zodpovedať pre každú vzorku malvéru tri základné otázky. Ako funguje, ako sa šíri a čo je jeho cieľom. Účelom je zozbierať dostatok informácií, ktoré je možné následne použiť naprieč celým spektrom bezpečnostno-obranných operácií.

Špecializované tímy zamerané na reakciu na bezpečnostné incidenty používajú túto metódu na rýchle posúdenie, či novoobjavený binárny súbor predstavuje okamžitú hrozbu, aké škody môže spôsobiť a aké sú najefektívnejšie spôsoby na zabránenie jeho šírenia alebo pre úplné odstránenie [6]. Dôležitosť takéhoto postupu je zrejmá.

Pohotová identifikácia rozsahu kompromitácie znižuje dobu zdržania (angl. *dwell time*[3]) a forenzná rekonštrukcia postupu útočníka umožňuje vytvoriť detailnú správu o incidente pre regulačné orgány, poisťovňu alebo interné potreby.

Okrem potreby okamžitej reakcie na incidenty má analýza malvéru aj dlhodobý strategický rozmer. Prináša praktické poznatky, ktoré pomáhajú pochopiť, ako útočníci operujú v širšom kontexte. Keď analytici napríklad dešifrujú konfiguračný súbor, extrahujú sieťový indikátor či obnovia šifrovací kľúč, získavajú informácie, ktoré objasňujú fungovanie konkrétnych rodín malvéru. Tieto zistenia sú zdokumentované a zdieľané na dôveryhodných platformách, ako sú komunitné fóra, verejné databázy a podobne. Napokon sa premietnu do antivírusových signatúr, YARA pravidiel, mapovania ATT&CK techník a ďalších obranných opatrení. Jediná analyzovaná vzorka tak môže posilniť obranné schopnosti na globálnom meradle.

1.3.1 Statická a dynamická analýza

Samotný proces analýzy malvéru možno vykonať bez spustenia vzorky (statická analýza) alebo počas jej behu v kontrolovaných podmienkach (dynamická analýza). Toto rozdelenie sa dá zjednodušene chápať ako kontrola zameraná na programový kód a kontrola zameraná na programové správanie, aj keď v praxi sa často prelínajú a vzájomne dopĺňajú.

Pri statickom rozbore skúmame predovšetkým [7]:

- **Signatúry súboru**: Špecifické postupnosti bajtov, ktoré identifikujú danú vzorku malvéru. Databázy takýchto signatúr sú bežne využívané antivírusovými programami na detekciu malvéru.
- **Kryptografické haše**: Unikátne odtlačky (napríklad vytvorené pomocou algoritmov SHA-256 alebo MD5) používané na porovnávanie vzoriek so známymi databázami malvéru.
- **Reťazce a kľúčové slová**: Konkrétne podozrivé reťazce či kľúčové slová zapísané vo vzorke. Ako príklad možno uviesť kľúče registra, IP a URL adresy, známky obfuskácie a mnohé iné.
- **Metadáta súboru**: Informácie ako sú časové značky, veľkosť súboru, informácie o autorovi a podrobnosti o verzii, môžu napovedať či nedošlo k podozrivým zmenám alebo pokusom o maskovanie.

[3] časová dĺžka počas, ktorej má útočník voľný prístup k systému, rozumie sa od začiatku útoku až po detekciu

- **Štruktúra a formát súboru**: Anomálie ako sú neštandardné sekcie, neplatné hlavičky alebo chybné štruktúry vo formátoch pre spustiteľné súbory (napríklad PE, ELF), ktoré naznačujú manipuláciu alebo skrytý kód.

Dynamický prístup sleduje správanie vzorky počas jej vykonávania, najmä [7]:

- **Systémové a API volania**: Počas behu programu sú zaznamenané všetky vykonané systémové a API volania. Okrem názvov samotných volaní je dôležité ich poradie, počet opakovaní, prípadne použité parametre či návratové hodnoty. Tieto zistenia nám vedia odhaliť nízko-úrovňové akcie ako sú napríklad operácie so súbormi, vytvorenie sieťového pripojenia, zmeny v ochrane pamäte alebo manipuláciu s právami a podobne. Vzhľadom na to, že tieto stopy sú veľmi informatívne a detailné, tvoria hlavnú množinu príznakov pre väčšinu behaviorálnych detektorov aj modelov strojového učenia.

- **Procesy a vlákna**: Pozorované je vytváranie a manipulácia s procesmi a vláknami. Cieľom je analyzovať priebeh vykonávania a snahy o obídenie detekcie použitím techník ako je *process hollowing* [8], APC zaraďovanie [9] a podobne.

- **Aktivitu v súborovom systéme**: Monitorovanie čítaní, zápisov, premenovaní, mazaní, manipulácií s časovými značkami a ukladanie ďalšieho škodlivého obsahu alebo konfiguračných súborov. Dokážeme takto určiť ako malvér v systéme pretrváva, mení alebo zbiera dáta a či na to využíva dočasné súbory.

- **Sieťová komunikácia**: Zaznamenáva DNS dopyty, TCP/UDP relácie, HTTP požiadavky a šifrovacie podania rúk (angl. *handshake*) s cieľom odhaliť C2 koncové body, kanály na zber dát a techniky obfuskácie protokolov, ako sú algoritmy na generovanie domén alebo falšovanie TLS odtlačkov.

- **Interakcie používateľa a UI**: Sleduje zachytávanie obrazovky, *keylogging*, umelé udalosti vstupu, monitorovanie schránky so zámerom odhaliť pokusy o krádež prihlasovacích dát, funkcie spajvéru alebo automatizáciu sociálneho inžinierstva.

2 Algoritmy strojového učenia

Schopnosť zaradiť vzorky malvéru do presných kategórií (podkapitola 1.2) hrá kľúčovú úlohu v tvorbe efektívnych obranných stratégií, stanovení priorít pri reakcii na bezpečnostné incidenty a v porozumení motivácií útočníkov. Na rozdiel od tradičnej binárnej detekcie malvéru, kde je úlohou rozlíšiť medzi neškodným softvérom a malvérom, klasifikácia malvéru na kategórie si vyžaduje jemnejšie rozhodovacie hranice a vyššiu vysvetliteľnosť (viac v kapitole 3). Obzvlášť to platí u modelov strojového učenia, ktoré sú trénované na príznakoch vytvorených z dynamickej analýzy. Tie síce poskytujú detailný popis správania, no zároveň sú spojené s vysokou dimenzionalitou a riedkymi, často značne zašumenými dátami.

V tejto kapitole si popíšeme tri rôzne algoritmy strojového učenia a to náhodný les, XGBoost a viacvrstvový perceptrón. Všetky tri prístupy dokážu vykonávať ako klasifikáciu, tak aj regresiu, nás však bude zaujímať iba klasifikácia. Pre každý algoritmus zhrnieme základný spôsob fungovania a uvedieme niektoré silné a slabé stránky.

2.1 Náhodný les

Náhodný les (angl. *random forest*, skr. RF) je súborová metóda, ktorá využíva techniku *bagging* (odvodené z angl. *bootstrap aggregating*) na vytváranie viacerých rozhodovacích stromov a ich následnú kombináciu [10][11]. Princípom tejto techniky je, že spojením viacerých slabých modelov, trénovaných na rôznych podmnožinách dát, možno vytvoriť silný a robustný predikčný model. V rámci baggingu sa z pôvodného trénovacieho datasetu náhodne (s opakovaním) vytvorí niekoľko podmnožín. Tieto podmnožiny nazývame *bootstrap* vzorky. Na každej bootstrap vzorke je následne nezávisle natrénovaný rozhodovací strom. Výsledná klasifikačná predikcia je potom určená väčšinovým hlasovaním všetkých stromov.

Ďalším kľúčovým aspektom náhodného lesa je proces nazývaný *feature bagging*. Pri vytváraní každého stromu sa pri výbere rozdelení v uzloch (angl. *node split*) uvažuje iba náhodne zvolená podmnožina príznakov. Tento dodatočný prvok náhodnosti znižuje koreláciu medzi jednotlivými stromami a zároveň aj riziko preučenia (angl. *overfitting*). Vďaka tomu je RF robustný voči šumu v dátach a dobre funguje aj v prípade, že nie všetky príznaky sú relevantné alebo informatívne.

Veľkou výhodou RF je jednoduchá implementácia a relatívne nenáročné ladenie. Počet hyperparametrov je v porovnaní s niektorými inými algoritmami oveľa menší a funguje dobre aj s predvolenými hodnotami [12]. Najdôležitejšie sú počet stromov, maximálna hĺbka stromov, počet príznakov použitých pri rozdeľovaní uzla, minimálny počet vzoriek pre rozdelenie uzla a minimálny počet vzoriek v uzloch. Tréningový proces

navyše umožňuje efektívne využitie paralelizácie. Jednotlivé rozhodovacie stromy sú od seba navzájom nezávislé a tým pádom ich je možné trénovať súčasne.

RF taktiež poskytuje informácie o dôležitosti príznakov. Tie sa najčastejšie odhadujú dvoma spôsobmi:

- **Priemerné zníženie nečistoty**[4]: Vychádza z vnútornej štruktúry stromov v náhodnom lese [10]. Počas trénovania stromu sa pri každom rozdelení uzla vyberie príznak, ktorý najviac znižuje určitú mieru nečistoty (napr. Giniho index). Následne sa vyhodnocuje, o koľko daný príznak priemerne znížil túto nečistotu vo všetkých stromoch. Čím je zníženie väčšie, tým je väčšia dôležitosť príznaku. Výhodou tejto metódy je nízka výpočtová náročnosť. Nevýhodou je však tendencia zvýhodňovať príznaky s väčším počtom možných hodnôt (napr. kategorické dáta s veľkým počtom tried).

- **Priemerné zníženie presnosti**[5]: Metóda založená na permutačnom testovaní [13]. Najprv je odhadnutá presnosť modelu na validačnej množine (alebo na OOB vzorke). Následne sa pre každý príznak náhodne zamiešajú jeho hodnoty a opäť sa vyhodnotí presnosť modelu. Rozdiel medzi pôvodnou a novou presnosťou určuje dôležitosť príznaku. Čím väčší pokles nastal, tým je príznak dôležitejší. Táto metóda je odolnejšia voči zaujatosti pre určité typy príznakov, je však výpočtovo náročnejšia.

Oba prístupy poskytujú cenný pohľad na to, ktoré príznaky model považuje za najviac informatívne. V praxi sa často používajú oba, pre získanie komplexnejšieho obrazu. Ide však iba o globálny pohľad a neumožňuje lokálnu interpretáciu.

Napriek svojej robustnosti má RF aj viaceré nevýhody. Nie je vždy ľahké odhadnúť vhodnú hĺbku stromov. Plytké stromy nemusia zachytiť dostatočne komplexné vzory v dátach. Zatiaľ čo príliš hlboké stromy riskujú preučenie na bootstrap vzorkách. Podobne dôležitá je aj voľba správneho počtu stromov. Príliš málo stromov môže viesť k vysokému rozptylu modelu. Nadmerne veľký počet zas zvyšuje pamäťové nároky a predlžuje čas potrebný na predikciu. Preto najmä v produkčných systémoch, kde je rýchla odozva obzvlášť dôležitá, môže byť tento aspekt rozhodujúci.

Ďalším obmedzením je nižšia interpretovateľnosť modelu. Aj keď je možné odvodiť globálnu dôležitosť príznakov, tak rozhodovanie jednotlivých stromov v súbore zostáva nejasné. Na rozdiel od jednoduchých modelov, ako sú napr. rozhodovacie stromy alebo lineárne regresie, náhodný les patrí medzi tzv. *čierne skrinky* (angl. *black box*). Tento pojem označuje algoritmy, ktorých vnútorné fungovanie je pre človeka ťažko pochopiteľné.

[4]angl. *Mean Decrease in Impurity (MDI)*
[5]angl. *Mean Decrease in Accuracy (MDA)*

Vieme povedať, aký je vstup a aký výstup, no samotná logika rozhodovania nám zostáva skrytá.

Z praktického hľadiska je RF vhodný najmä pre štruktúrované dáta, kde vstupné príznaky môžu byť rôzneho typu, číselné aj kategorizované. Nie je citlivý na škálovanie vstupov a dobre si poradí aj s dátami, ktoré obsahujú chýbajúce hodnoty alebo irelevantné príznaky. Hoci je dnes často nahradený novšími a výkonnejšími metódami ako XGBoost alebo hlboké neurónové siete, naďalej zostáva užitočným *baseline* modelom pre klasifikačné úlohy [14].

2.2 XGBoost

Extreme Gradient Boosting (skr. XGBoost alebo XGB) je súborová metóda, ktorá vytvára sériu rozhodovacích stromov sekvenčným spôsobom, pričom každý nový strom je trénovaný tak, aby napravil chyby aktuálneho modelu [15][16]. Tento prístup patrí do širšej rodiny tzv. *boosting* algoritmov. Jedná sa o súborové metódy, ktoré vytvárajú silný model postupným pridávaním viacerých slabých modelov jeden za druhým. Základnou myšlienkou boostingu je *aditivita*. Začína sa s modelom, ktorý má len mierne lepšiu výkonnosť než náhodný odhad. Každý nový model je trénovaný na zvyškových chybách[6], ktoré zostali po doterajšom súborovom modeli. V XGBooste je každý nový strom prispôsobený prvým a druhým deriváciám (gradientom a Hessianom) diferencovateľnej chybovej funkcie, čím je dosiahnutá ešte presnejšia aktualizácia. Pri binárnej klasifikácii sa typicky zameriava na minimalizáciu regularizovanej logistickej chyby, zatiaľ čo pri úlohách s viacerými triedami sa sústredí na minimalizáciu generalizačnej funkcie softmax. V iterácii m sa aktuálna predikcia $\hat{y}^{(m-1)}$ prevedie na pravdepodobnosti tried a nový strom f_m sa prispôsobí vypočítaným gradientom a Hessianom. Prínos z rozdelenia uzla je odhadnutý na základe týchto derivácií. Algoritmus teda vie dopredu, o koľko by každý kandidát na rozdelenie uzla znížil celkovú chybu. Keď je nájdená optimálna štruktúra stromu, váhy listov sú upravené pomocou faktora rýchlosti učenia η. Výstup zo súborového modelu sa následne upraví podľa vzorca:

$$\hat{y}^{(m)} = \hat{y}^{(m-1)} + \eta f_m(x)$$

Tento aditívny, gradientom riadený proces pokračuje, kým nie je dosiahnutý zvolený počet boosting iterácií alebo kým sa nezastaví zlepšovanie na validačnej množine, v prípade použitej techniky skoršieho zastavenia. Na rozdiel od AdaBoostu, ktorý priraďuje rôzne váhy jednotlivým vzorkám v závislosti od ich predchádzajúcich chýb, XGBoost optimalizuje priamo v priestore funkcií pomocou gradientnej informácie. Vďaka čomu je flexibilnejší a efektívnejší.

[6] vo všeobecnosti na zápornom gradiente chyby

Na zabránenie preučeniu na trénovacích dátach sú použité dve formy regularizácie. Ako prvé, cieľová funkcia obsahuje priame penalizácie. Každý ďalší list je penalizovaný fixnou hodnotou a veľké hodnoty váh listov sú penalizované kvadraticky. Po druhé, rýchlosť učenia[7] a náhodné podvzorkovanie (angl. *subsampling*) riadkov a stĺpcov prinášajú do procesu dodatočný šum, ktorý zabezpečuje mierne odchýlky na trénovacích dátach pre jednotlivé stromy. Dokopy, tieto mechanizmy pripomínajú efekt redukcie rozptylu, tak ako ho poznáme z baggingu. Zároveň si však zachovávajú nízku dátovú zaujatosť (angl. *bias*), ktorou je boosting známy.

Z implementačného hľadiska XGBoost uchováva príznaky v blokovo komprimovanom formáte, uchováva štatistiky gradientu v súvislej pamäti a spolieha sa na viacvláknové histogramy či GPU jadrá na vyhodnocovanie rozdelení. Vďaka tomu je škálovateľný takmer lineárne s počtom pozorovaní a dokáže spracovať milióny vzoriek so stovkami príznakov aj na bežnom hardvéri.

Keďže každý strom je zameraný priamo na gradienty klasifikačnej chyby, celkový súbor stromov má tendenciu konvergovať k veľmi presnej rozhodovacej hranici. Navyše na to potrebuje menší počet stromov než klasický AdaBoost alebo jednoduchý gradientový boosting. Voľbou vhodných hyperparametrov ako sú faktor rýchlosti učenia, maximálna hĺbka stromu, minimálna váha dieťaťa alebo pomer podvzorkovania umožňujú vymeniť menší pokles v trénovacej presnosti za výraznú redukciu v preučení. XGBoost si našiel široké uplatnenie vďaka svojej efektivite a schopnosti dosahovať výborné výsledky. Bol použitý mnohými víťaznými tímami v súťažiach strojového učenia [17].

Tak ako v prípade náhodného lesa, aj XGBoost poskytuje rôzne spôsoby vyhodnotenia dôležitosti jednotlivých príznakov. Poskytovaných je päť hlavných metrík [18]:

- **Váha** (angl. *weight*, tiež frekvencia): Udáva, koľkokrát bol daný príznak použitý na rozdelenie uzla v priebehu trénovania všetkých stromov. Ide o jednoduchú metriku, ktorá však môže byť zavádzajúca. Nezohľadňuje totiž kvalitu týchto rozdelení a ani typ príznaku.

- **Prínos** (angl. *gain*): Meria priemerný prínos daného príznaku na zlepšení cieľovej funkcie (o koľko sa pri rozdelení uzla znížila chyba). Základom výpočtu prínosu je rozdiel v hodnote optimalizačnej funkcie pred a po rozdelení, pričom sa využívajú gradienty a Hessiany. Príznaky, ktoré najviac znížili chybu, získajú najvyššie skóre. Táto metrika je analogická k priemernému zníženiu nečistoty (MDI) z náhodného lesa. V XGBooste je však presnejšia, keďže vychádza z derivácií skutočnej chybovej funkcie.

[7] nazývané aj zmenšovanie (angl. *shrinkage*) alebo η (eta)

- **Celkový prínos** (angl. *total gain*): Sčítanie všetkých prínosov daného príznaku naprieč všetkými stromami. Často poskytuje presnejší obraz o kumulatívnej užitočnosti príznaku v rámci celého modelu, najmä pri vizualizáciách dôležitosti.
- **Pokrytie** (angl. *cover*): Určuje, koľko vzoriek bolo rozdelením ovplyvnených pri použití konkrétneho príznaku. V prípade vážených dát je namiesto počtu vzoriek metrika definovaná ako súčet váh.
- **Celkové pokrytie** (angl. *total cover*): Súčet pokrytia pre všetky výskyty príznaku v rozhodovacích stromoch. Pomáha zhodnotiť rozsah vplyvu príznaku, nezávisle od toho, aký veľký bol prínos k zníženiu chyby.

Okrem týchto vstavaných metrík je možné využiť aj permutačné testovanie. To je totožné ako priemerné zníženie presnosti (MDA) pre náhodný les. Tieto metriky umožňujú získať globálny pohľad na správanie modelu.

Použitie XGBoostu prináša aj mnohé výzvy. Ladenie modelov je obzvlášť náročné pre menej skúsených používateľov. Desiatky hyperparametrov sa navzájom ovplyvňujú mnohými, na prvý pohľad nejasnými, spôsobmi. Predvolené nastavenia, ktoré fungujú pre jeden klasifikačný problém, môžu na inom viesť k nízkej presnosti alebo k preučeniu. Keďže boosting je zo svojej podstaty sekvenčný, trénovanie nemožno paralelizovať tak jednoducho ako pri modeloch náhodného lesa. Pokiaľ je dôležitá rýchlosť odozvy, potrebné je dať pozor aj na počet stromov, keďže čas potrebný na predikciu s ním rastie lineárne. Veľké stromové súbory, najmä tie s hlbokými stromami, môžu zaberať stovky MB pamäte, čo môže predstavovať problém pre menej výkonné zariadenia.

Z hľadiska interpretovateľnosti patrí XGBoost, podobne ako náhodný les, do skupiny čiernych skriniek. Hoci rozhodovacie stromy, z ktorých sa XGBoost skladá, sú jednotlivo pomerne ľahko interpretovateľné, ich kombinácia v rámci súboru vytvára zložité a neprehľadné rozhodovacie štruktúry. Je síce možné získať určité formy globálneho vyhodnotenia dôležitosti príznakov, tieto metriky však nevysvetľujú, ako konkrétne vstupy ovplyvňujú jednotlivé predikcie.

2.3 Viacvrstvový perceptrón

Viacvrstvový perceptrón (angl. *multilayer perceptron*, skr. MLP) je najznámejším predstaviteľom doprednej neurónovej siete a jedným zo základných modelov strojového učenia [19]. Zjednodušene, premieňa vstupný vektor príznakov $x \in \mathbb{R}^d$ na predikciu $\hat{y}$. Na tento účel využíva postupnosti plne prepojených vrstiev, ktoré sú tvorené z lineárnych transformácií a za nimi nasledujúcimi nelineárnymi aktivačnými funkciami. Typ úlohy určuje špecifickú funkciu výstupnej vrstvy. Pre binárnu klasifikáciu ide najčastejšie o

sigmoidu a pre viactriednu klasifikáciu zas o softmax. Výsledkom sú pravdepodobnosti, z ktorých sieť odvodzuje konečnú triedu.

Základným procesom každej doprednej siete je dopredné šírenie. Formálne si ho môžeme definovať ako:

$$
\begin{aligned}
a_0 &= x \\
z_l &= W_l a_{l-1} + b_l \quad (l = 1, \ldots, L) \\
a_l &= \phi_l(z_l) \quad (l = 1, \ldots, L-1) \\
\hat{y} &= g(z_L)
\end{aligned}
$$

Kde W_l a b_l predstavujú váhovú maticu a bias (vektor posunu) pre vrstvu l. ϕ_l je aktivačná funkcia danej skrytej vrstvy (napr. ReLU alebo ELU) a g je funkcia, ktorá transformuje výstup poslednej vrstvy do finálnej predikcie (napr. softmax). Každý neurón v danej vrstve je prepojený so všetkými neurónmi z predchádzajúcej vrstvy, čo umožňuje modelu zachytávať aj komplexné vzťahy medzi vstupnými príznakmi. Vďaka tomu je MLP vhodným nástrojom pre riešenie klasifikačných úloh s komplikovanými a spojitými vstupmi.

Tréning na klasifikáciu je založený na minimalizovaní krížovej entropie chyby medzi predikovanými predpoveďami a skutočnými triedami v datasete. Toto je zvyčajne dosiahnuté použitím optimalizačných metód založených na gradientoch (napr. SGD alebo Adam). Gradienty sú spočítané cez algoritmus spätného šírenia chyby (angl. *backpropagation*). Ten využíva reťazové pravidlo na postupné šírenie derivácií od výstupnej vrstvy smerom dozadu cez sieť. Tieto výpočty, založené najmä na hustých maticových operáciách, sú vhodné na paralelizáciu použitím moderných GPU. Vďaka tomu je možné efektívne trénovať MLP modely s miliónmi parametrov aj na veľkých datasetoch.

Komplexné modely sú však náchylné na preučenie, najmä ak nie je trénovacia množina dostatočne veľká. Na zmiernenie tohto rizika sa využíva viacero regularizačných stratégií:

- **Dropout** je technika, pri ktorej sa počas tréningu náhodne vypína časť neurónov. Cieľom je znížiť závislosť modelu na konkrétnych neurónoch a zvýšiť odolnosť voči šumu na dátach.
- **Dávková normalizácia** (angl. *batch normalization* zabezpečuje počas tréningu normalizovanie vstupov do každej vrstvy tak, aby mali konzistentné štatistické vlastnosti (napr. strednú hodnotu a rozptyl). Tým sa zrýchľuje a stabilizuje proces učenia.
- **Rozklad váh** (angl. *weight decay*, alebo L2 regularizácia) pridáva k chybovej funkcií regulačný člen penalizujúci príliš veľké váhové hodnoty. Zámerom je

prinútiť model uprednostňovať jednoduchšie riešenia s jemnejšími rozhodovacími hranicami.

- **Skoré zastavenie** (angl. *early stopping*) je technika, ktorá počas tréningu priebežne sleduje výkon modelu na validačnej množine. V prípade, že sa vopred stanovený počet epoch nezlepší, tréning sa ukončí. Cieľom je zachovať generalizačnú schopnosť modelu.

Dôležitú úlohu zohráva aj výber aktivačnej funkcie [20]. Staršie MLP siete využívali sigmoid alebo tanh funkcie, ktoré však trpia problémom miznúcich gradientov. Pri väčších vstupoch sa ich derivácie blížia k nule, čo znemožňovalo spätné šírenie gradientu sieťou. Moderné siete využívajú tzv. „po častiach lineárne“ (angl. *piecewise linear*) funkcie, akými sú ReLU, Leaky ReLU alebo ELU. Ich výhodou je, že pri väčšine vstupov nespôsobujú saturáciu výstupu ani prudké zmenšovanie derivácie. Vďaka tomu umožňujú efektívnejší prenos gradientu počas učenia. Spoločne s vhodnou váhovou inicializačnou schémou (napr. He alebo Xavier) si tieto aktivačné funkcie dokážu udržať stabilnú veľkosť signálu a gradientov naprieč vrstvami, čo umožňuje úspešné trénovanie aj hlbších sietí.

Napriek svojej flexibilite sú MLP siete citlivé na vhodný výber hyperparametrov. Sem patrí napr. počet skrytých vrstiev, šírka vrstiev (počet neurónov), faktor rýchlosti učenia, sila regularizácie či počet vzoriek spracovaných v jednej dávke (angl. *batch*). Vyžadujú si dôkladné ladenie, keďže na seba navzájom rôzne, často nepredvídateľne, vplývajú.

Veľkou nevýhodou MLP modelov je ich nízka interpretovateľnosť. Rovnako ako predchádzajúce súborové metódy, aj neurónové siete sa zaraďujú medzi modely čiernej skrinky. Správanie jedného neurónu je síce relatívne ľahké na pochopenie, keďže vykonáva pomerne jednoduché lineárne transformácie nasledované nelineárnou aktivačnou funkciou. Výsledná predikcia však vzniká ako kombinácia veľkého počtu vrstiev a neurónov, ktorých vzájomné interakcie vytvárajú komplexné rozhodovacie štruktúry. Toto robí ich pochopenie pre človeka prakticky nemožným.

3 Vysvetliteľnosť v strojovom učení

Strojové učenie sa stalo neoddeliteľnou súčasťou moderných technológií a dnes je už ťažké si predstaviť fungovanie mnohých odvetví bez jeho využitia. Výrazné zlepšenia v efektivite aj presnosti prinieslo najmä v oblastiach, ako napr. klasifikácia obrázkov, spracovanie prirodzeného jazyka, autonómne riadenie, kybernetická bezpečnosť či medicína.

Napriek týmto úspechom však väčšina populárnych modelov čelí zásadnému problému. Sú to čierne skrinky, ktorých fungovanie nie je zvonku viditeľné ani zrozumiteľné. Táto netransparentnosť vyvoláva oprávnené obavy o dôveryhodnosť, férovosť a tiež sociálnu akceptáciu v spoločnosti [21]. Tieto obavy sú o to závažnejšie v kritických oblastiach, akými sú zdravotníctvo či kybernetická bezpečnosť.

Rovnakým výzvam čelia aj algoritmy RF, XGBoost a MLP, ktoré sme predstavili v predchádzajúcej kapitole. Rozhodnutia, ktoré tieto algoritmy robia, sú síce často veľmi presné, no zároveň často pre človeka nepochopiteľné. Tento nedostatok prehľadnosti vyvoláva množstvo otázok: Môžeme takémuto modelu dôverovať? Je jeho rozhodovanie spravodlivé? Ako dospel k danému výsledku?

Na tieto výzvy odpovedá oblasť známa ako vysvetliteľná umelá inteligencia (angl. *explainable artificial intelligence*, skrátene XAI). Jej cieľom je vývoj nástrojov a techník, ktoré umožnia lepšie porozumenie komplexným modelom a to bez toho, aby sa znížil ich výkon.

Poznámka: Keď hovoríme o vysvetľovaní modelov strojového učenia, konkrétnejšie ide o oblasť vysvetliteľného strojového učenia (angl. explainable machine learning). Samotný pojem XAI zahŕňa aj širšie aspekty umelej inteligencie, ako napr. symbolické systémy. Častejšie sa však v praxi tento termín používa aj v súvislosti s vysvetľovaním modelov strojového učenia. Preto ho v tomto význame používame aj v tejto práci.

3.1 Základné koncepty v XAI

Oblasť XAI zahŕňa množstvo konceptov, ktoré sú nevyhnutné pre správne pochopenie princípov vysvetľovania modelov strojového učenia. Vzhľadom na rozmanitosť prístupov a miestami nejednotnú terminológiu v odbornej literatúre, môže byť orientácia v tejto téme náročná, obzvlášť pre čitateľov bez technickej prípravy v tejto oblasti.

Táto podkapitola preto poskytuje prehľad kľúčových základných pojmov, ktoré sú potrebné pre porozumenie ďalším častiam tejto práce. Začneme s rozdielmi medzi často zamieňanými pojmami „vysvetliteľnosť“ a „interpretovateľnosť“ v časti 3.1.1. Následne si v časti 3.1.2 klasifikujeme vysvetľovacie metódy podľa ich vzťahu k modelu. Nakoniec

sa v časti 3.1.3 zameriame na výstup z týchto metód, konkrétne na rozdiel medzi lokálnymi a globálnymi vysvetleniami.

3.1.1 Vysvetliteľnosť a interpretovateľnosť

Pojmy vysvetliteľnosť (angl. *explainability*) a interpretovateľnosť (angl. *interpretability*) sa v literatúre často zamieňajú, predstavujú však dva príbuzné, no odlišné koncepty. V rámci XAI komunity navyše neexistuje jednotná a univerzálne prijatá definícia týchto pojmov, čo spôsobuje ďalšie nejasnosti [22]. Napriek tomu si ich v tejto práci zadefinujeme nasledovne [23]:

- **Interpretovateľnosť**: Označuje mieru, do akej človek dokáže priamo pochopiť mechanizmus fungovania modelu a jeho rozhodovania. Interpretovateľný model umožňuje používateľovi identifikovať vzťahy medzi vstupnými dátami a výstupnými predikciami. Pritom musí byť zrejmé, akým spôsobom sú vstupné príznaky použité pri vytváraní výsledku. Medzi dobre interpretovateľné modely patrí napr. lineárna regresia, logistická regresia alebo jednoduché rozhodovacie stromy.
- **Vysvetliteľnosť**: Predstavuje schopnosť modelu poskytnúť zrozumiteľné odôvodnenia svojich rozhodnutí, väčšinou však až dodatočne po vykonaní predikcie. Takéto vysvetlenia sú často generované pomocou externých metód. Tie môžu byť aplikované aj na zložité, neinterpretovateľné modely, ako sú napr. hlboké neurónové siete alebo súborové modely.

Treba však zdôrazniť, že nejde o dva úplne oddelené koncepty. Naopak, vysvetliteľnosť a interpretovateľnosť možno vnímať ako body na kontinuálnom spektre. Na jednej strane stoja plne transparentné a ľahko pochopiteľné modely, ktoré sú zo svojej podstaty interpretovateľné. Na opačnom konci sa nachádzajú komplexné modely, ktorých rozhodovanie je pre človeka, bez dodatočných vysvetlení, absolútne nepochopiteľné. Medzi týmito extrémami existuje množstvo modelov a prístupov, ktoré môžu byť viac či menej interpretovateľné alebo vysvetliteľné v závislosti od kontextu, použitých metód a cieľovej skupiny.

3.1.2 Typy vysvetľovacích metód podľa vzťahu k modelu

Ako sme naznačili v predchádzajúcej časti, modely strojového učenia sa líšia v miere zrozumiteľnosti, a preto si vyžadujú rôzne prístupy k vysvetľovaniu svojich rozhodnutí. Jedným zo spôsobov na kategorizáciu vysvetľovacích metód patrí ich vzťah ku konkrétnemu typu modelu. Tu rozlišujeme dve kategórie [23]:

- **Modelovo-špecifické** (angl. *model-specific*) metódy sú úzko späté s konkrétnou architektúrou alebo algoritmom modelu. Na generovanie vysvetlení využívajú špecifické vnútorne vlastnosti modelu. Príkladom je DeepLIFT, integrované gradienty alebo merania dôležitosti príznakov pre náhodný les či XGBoost. Výhodou týchto prístupov je ich zvyčajne vyššia presnosť a schopnosť poskytnúť detailnejší pohľad do rozhodovacieho procesu modelu. Ich použiteľnosť je však obmedzená, pretože sú aplikovateľné iba pre modely, pre ktoré boli navrhnuté.

- **Modelovo-agnostické** (angl. *model-agnostic*) metódy fungujú nezávisle od konkrétneho typu modelu. Tieto techniky pracujú s modelom ako s čiernou skrinkou. To znamená, že vysvetlenia generujú na základe analýzy výstupov modelu pri rôznych vstupoch. Známymi predstaviteľmi sú metódy ako SHAP (Kernel variant), Anchors alebo rôzne implementácie kontrafaktuálov. Ich hlavnou výhodou je univerzálnosť, keďže umožňujú získať vysvetlenia pre rôzne modely bez nutnosti poznať ich vnútornú štruktúru. Na druhej strane sú však zvyčajne výpočtovo náročnejšie a nemusia poskytnúť také presné vysvetlenia ako metódy, ktoré majú prístup k interným parametrom modelu.

Voľba medzi modelovo-špecifickými a modelovo-agnostickými metódami závisí od účelu použitia. Špecifickejšie metódy sú zvyčajne presnejšie a rýchlejšie, no nie sú vždy dostupné pre každý model a často sa zameriavajú len na určitý typ vysvetlení (napr. globálne). V iných prípadoch, najmä ak potrebujeme pracovať s rozličnými modelmi naraz alebo špecifické metódy nespĺňajú naše požiadavky, sú vhodnejšie agnostické prístupy.

3.1.3 Lokálne a globálne vysvetlenia

Analogicky k rozdeleniu metód vysvetliteľnosti, môžeme rozlíšiť aj typy výstupov, ktoré tieto metódy vyprodukujú. Rozlišujeme dva základné typy vysvetlení [24]:

- **Lokálne**: Sústreďujú sa na vysvetlenie individuálnych predikcií modelu. Poskytujú náhľad do dôvodov, prečo model v konkrétnom prípade urobil určité rozhodnutie. Umožňujú identifikovať, ktoré vstupné znaky najvýraznejšie ovplyvnili (a ako) výsledok pre konkrétny vstup. Vďaka čomu zvyšujú dôveru používateľa v rozhodovanie modelu.

- **Globálne**: Zameriavajú sa na objasnenie celkového správania modelu. Cieľom je zachytiť všeobecné vzory a logiku, ktorú sa model naučil počas trénovania. Globálne vysvetlenia sú obzvlášť prínosné pri overovaní súladu modelu s odbornými znalosťami, pri odhaľovaní zaujatosti alebo pri získavaní prehľadu o dôležitosti jednotlivých príznakov (napr. podľa priemeru alebo absolútneho súčtu).

Nie je však potrebné sa obmedzovať len na jeden typ vysvetlení. Pre lepšie porozumenie správania modelu je ideálne kombinovať lokálne aj globálne prístupy, keďže každý z nich poskytuje jedinečný pohľad na rozhodovanie modelu. Zatiaľ čo globálne vysvetlenia je možné použiť na pochopenie všeobecných princípov, lokálne vysvetlenia ponúkajú detailný pohľad na konkrétne predikcie. Väčšina vysvetľovacích metód však podporuje len jeden z týchto prístupov. Dokážu teda vytvárať buď lokálne, alebo globálne vysvetlenia. Preto je často potrebné kombinovať viacero nástrojov. Existujú však aj univerzálne prístupy (napr. SHAP), ktoré umožňujú tvorbu oboch typov vysvetlení.

3.2 SHAP

Shapley Additive Explanations (SHAP) je súbor modelovo-agnostických aj modelovo-špecifických vysvetľovacích metód, ktorý umožňuje vytvárať lokálne aj globálne vysvetlenia [23][25]. Vychádza z teórie kooperatívnych hier a jej výstupom sú tzv. „SHAP hodnoty“, ktoré pre každý príznak vyjadrujú prínos k predikcii modelu pre konkrétnu vzorku. Ich výpočet je postavený na aproximácii Shapleyho hodnôt, ktoré boli pôvodne vyvinuté na férové rozdelenie zisku medzi hráčmi, v závislosti od ich celkového prínosu. Analogicky, pri aplikovaní na predikciu modelov strojového učenia, hráči predstavujú jednotlivé príznaky[8] a ziskom je podiel na výstupe z modelu. Shapleyho hodnota potom pre každý príznak zodpovedá jeho priemernému marginálnemu príspevku[9] naprieč všetkými možnými podmnožinami príznakov.

Presnejšie, Shapleyho hodnoty sa počítajú zohľadnením všetkých možných podmnožín vstupných príznakov, kde pre každú podmnožinu, ktorá neobsahuje daný príznak, je vypočítaný marginálny príspevok pridania tohto príznaku [23]. Shapleyho hodnota príznaku je potom váženým priemerom týchto marginálnych príspevkov cez všetky podmnožiny. Formálne je Shapleyho hodnota ϕ_i pre príznak i definovaná ako:

$$\phi_i(f) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{i\}} \frac{|S|!(p - |S| - 1)!}{p!} \left[f(S \cup \{i\}) - f(S) \right]$$

Kde S je podmnožina príznakov, p je počet príznakov a $f(X)$ je hodnotová funkcia, ktorá vyjadruje výstup modelu, ak sú známe iba príznaky v množine X.

Shapleyho hodnoty sú jediným spôsobom rozdelenia podľa zásluh medzi účastníkmi, ktorý spĺňa tzv. axiómy spravodlivosti. Ich splnenie je považované za definíciu spravodlivého rozdelenia. V kontexte vysvetlenia predikcie modelu, to sú:

[8] konkrétnejšie, hodnoty jednotlivých príznakov

[9] angl. *marginal contribution*, určuje akú dodatočnú hodnotu prinesie, keď sa pridá do podmnožiny

- **Efektivita** – súčet zásluh príznakov na predikcii sa rovná rozdielu predikcie od priemernej (očakávanej) hodnoty

$$\sum_{i=1}^{p} \phi_i = \hat{f}(\mathbf{x}) - \mathbb{E}[\hat{f}(\mathbf{X})]$$

- **Symetria** – príznaky s rovnakým podielom na predikcii majú rovnakú zásluhu na predikcii

$$\text{if} \quad f(\mathcal{S} \cup \{i\}) = f(\mathcal{S} \cup \{j\}) \quad \text{for all} \quad \mathcal{S} \subseteq \{1, \ldots, p\} \setminus \{i, j\} \quad \text{then} \quad \phi_i = \phi_j$$

- **Nulový hráč** – príznak neovplyvňujúci predikciu má nulovú zásluhu

$$\text{if} \quad f(\mathcal{S} \cup \{i\}) = f(\mathcal{S}) \quad \text{for all} \quad \mathcal{S} \subseteq \{1, \ldots, p\} \quad \text{then} \quad \phi_i = 0$$

- **Aditivita** – pre súborové modely, sú zásluhy súboru rovné súčtu zásluh komponentov súboru

V praxi je však pre modely s väčším množstvom príznakov výpočet presných Shapleyho hodnôt výpočtovo neuskutočniteľný. Vyžaduje totiž model natrénovať 2^p-krát, kde p je počet príznakov. V dôsledku toho boli vyvinuté rôzne aproximačné a špecializované algoritmy. Jedným z takýchto algoritmov je aj SHAP. K dispozícii je viacero rôznych variácií, pričom každý je optimalizovaný pre špecifický prípad.

3.2.1 KernelSHAP

KernelSHAP je modelovo-agnostický variant, ktorý odhaduje Shapleyho hodnoty tým, že s každým príznakom zaobchádza ako s prítomným alebo chýbajúcim v predikcii [23][25][26]. Aby simuloval, že je príznak „chýbajúci“, algoritmus jeho hodnotu nahradí hodnotami naprieč viacerými riadkami z dodaného podkladového datasetu. Potom spriemeruje predikcie modelu pre všetky nahradené hodnoty. Týmto postupom odhadne, čo by model predpovedal, ak by nepoznal daný príznak. Algoritmus potom vzorkuje kombinácie binárnych masiek (koalície), ktoré indikujú, ktoré príznaky sú prítomné. Masky s veľmi malým alebo naopak takmer úplným počtom príznakov sa vyberajú častejšie, keďže poskytujú najviac informácií. Pre každú vzorkovanú masku je zaznamenaný výstup z modelu. Následne sa vytvorí vážený lineárny regresný model, kde vstupy sú masky a cieľom sú zaznamenané predikcie modelu. Koeficienty príznakov z regresie sú potom konečným odhadom Shapleyho hodnôt.

KernelSHAP dobre odhaduje skutočné Shapleyho hodnoty, čím zaručuje splnenie axiómov spravodlivosti. Okrem toho, je základná myšlienka, spriemerovanie výstupov modelu naprieč rôznymi kombináciami známych a neznámych príznakov, pomerne

jednoduchá a intuitívna. Má však množstvo praktických nevýhod, kvôli ktorým sa už zvyčajne neodporúča. Aj napriek tomu, že je v porovnaní s klasickým výpočtom Shapleyho hodnôt oveľa efektívnejší, pri väčšom množstve príznakov je stále pomerne výpočtovo náročný. Získanie presných odhadov si totiž vyžaduje veľké množstvo predikcií. Metóda môže tiež vytvárať šumové vysvetlenia, ak sa použije príliš málo vzoriek, keďže sa spolieha na náhodné vzorkovanie a regresný model. KernelSHAP takisto predpokladá nezávislosť príznakov pri simulovaní chýbajúcich hodnôt. V reálnych dátach, kde sú príznaky často korelované, môže tento predpoklad viesť k zavádzajúcim alebo nespoľahlivým vysvetleniam.

3.2.2 TreeSHAP

TreeSHAP je modelovo-špecifický variant určený pre rozhodovacie stromy a súborové modely založené na stromoch, ako je napríklad náhodný les alebo XGBoost [23][27][28]. Namiesto náhodného simulovania chýbajúcich príznakov a opätovného vyhodnocovania modelu využíva vnútornú štruktúru stromov na presný výpočet prínosu každého príznaku v polynomiálnom čase. Algoritmus prechádza každú cestu od koreňa stromu k listom a pri každom rozdelení uzla počíta pravdepodobnosť, že náhodne vybraná podmnožina príznakov bude nasledovať danú vetvu. Táto pravdepodobnosť závisí od toho, či sa rozdeľujúci príznak nachádza v podmnožine (rozhodnutie je vtedy deterministické) alebo chýba (rozdelenie sa urobí proporcionálne podľa štatistiky z trénovacích dát). Ako algoritmus postupuje, kumuluje pravdepodobnosti jednotlivých rozdelení. Po dosiahnutí listu je predikcia listu vážená celkovou pravdepodobnosťou danej cesty. Spojením všetkých ciest dosiahne algoritmus celkový prínos všetkých možných podmnožín príznakov aj bez potreby priameho vyčíslenia všetkých 2^p možností. Získava tak presné Shapleyho hodnoty pre výstup modelu. V prípade súborových modelov je tento postup aplikovaný nezávisle na každom strome. Prínosy jednotlivých príznakov sú vypočítané pre každý strom a nakoniec sčítané pre celý súborový model. Odráža tým spôsob, akým model kombinuje svoje predikcie.

Výpočtová náročnosť pre TreeSHAP je v porovnaní s KernelSHAP znížená z $O(TL2^p)$ na $O(TLD^2)$, kde T je počet stromov, L je maximálny počet listov a D je maximálna hĺbka. Okrem toho je algoritmus deterministický, takže nevzniká šum zo vzorkovania, ktorý je typický pre KernelSHAP. Na rozdiel od maskovacích metód, nepredpokladá nezávislosť medzi príznakmi. Napríklad, ak sú dva príznaky korelované, strom to prirodzene zachytí tým, že rozdelí uzol na druhom príznaku len v prípade, že pridáva novú informáciu. TreeSHAP zachytí túto sekvenciu rozdelení, vďaka čomu vernejšie odráža skutočnú závislosť medzi príznakmi. TreeSHAP však nie je bez obmedzení. Správnosť výpočtu je garantovaná iba pre čisto stromové modely. Ak predikčná pipeline

obsahuje ľubovoľné predspracovanie či iný model ako stromový, nie je TreeSHAP možné použiť. Okrem toho môže byť spotreba pamäte výrazná, najmä pri použití súborových modelov s viac ako miliónom listov. A keďže výpočtová náročnosť v najhoršom prípade rastie kvadraticky s hĺbkou stromu, veľmi hlboké stromy môžu viesť k dlhým časom spracovania.

3.2.3 DeepSHAP

DeepSHAP je modelovo-špecifický variant určený pre neurónové siete, ktorý kombinuje prístupy SHAP a DeepLIFT. Samotný DeepLIFT je vysvetľovacia metóda pre neurónové siete, založená na porovnávaní aktivácií neurónov s ich referenčnými aktiváciami [29]. Rozdiely medzi skutočnými a referenčnými aktiváciami sa spätne propagujú sieťou podľa stanovených pravidiel, čím sa určuje prínos jednotlivých neurónov. Referenčné aktivácie zodpovedajú stavom neurónov pri spracovaní referenčného vstupu. Ten je zvyčajne zvolený tak, aby predstavoval „neutrálny“ stav, napríklad všetky nuly, priemer trénovacích dát alebo náhodnú vzorku z podkladového datasetu. Rozdiely voči týmto hodnotám vyjadrujú odchýlku od základnej úrovne a slúžia na výpočet významnosti príznakov.

DeepSHAP tento princíp rozširuje s cieľom lepšie spĺňať teoretické vlastnosti SHAP-u, ako sú lokálna presnosť a konzistencia [25][30]. Na rozdiel od klasického DeepLIFT, ktorý sa spolieha na jeden referenčný vstup, DeepSHAP používa viacero vzoriek z podkladového datasetu, čím zlepšuje odhad očakávaní nad chýbajúcimi príznakmi. Pri vysvetľovaní DeepSHAP rozkladá sieť na jednoduché operácie, ako sú lineárne transformácie, aktivačné funkcie a pooling operácie. Pre každú z nich počíta lokálne SHAP hodnoty buď presným riešením, alebo použitím lokálnych lineárnych aproximácií, v závislosti od charakteru operácie. Lokálne SHAP hodnoty zodpovedajú tomu, ako vstupy každej operácie prispievajú k jej výstupu. Nakoniec DeepSHAP aplikuje modifikované pravidlo reťazenia. Lokálne príspevky sú spätne propagované vrstva za vrstvou od výstupu siete až po jej vstupy. V každom kroku sa zásluhy rozdeľujú medzi predchádzajúce vrstvy na základe vypočítaných lokálnych príspevkov. Tento proces pokračuje, až kým všetky zásluhy nie sú vyjadrené výhradne pomocou pôvodných vstupných príznakov. Výsledkom je priradenie SHAP hodnôt každému vstupnému príznaku, ktoré vyjadrujú jeho odhadovaný prínos na predikcii modelu.

DeepSHAP si zachováva hlavné silné stránky SHAP a DeepLIFT. Poskytuje verné a konzistentné prínosy príznakov a dobre zachytáva aj nelineárne vzťahy. Čas spracovania rastie približne lineárne s veľkosťou siete (počtom vrstiev a neurónov), vďaka čomu je DeepSHAP rádovo rýchlejší než KernelSHAP. Na rozdiel od čisto gradientových metód sa vyhýba problémom saturácie (kde gradienty miznú aj keď sú príznaky dôležité),

pretože je založený na konečných diferenciách namiesto lokálnych derivácií. DeepSHAP však preberá aj niektoré obmedzenia. Veľkosť podkladového datasetu priamo ovplyvňuje kvalitu odhadu, no s väčším počtom vzoriek výrazne rastie aj výpočtová náročnosť. DeepSHAP, podobne ako KernelSHAP, predpokladá nezávislosť príznakov. Takže ak existuje silná korelácia medzi príznakmi v trénovacích dátach, nemusí byť odhad presný. Navyše architektúry, ktoré narúšajú jednoduchý dopredný výpočet, ako sú napr. stochastické vrstvy, dynamické riadenie toku alebo určité normalizačné schémy, nie sú podporované.

3.2.4 Globálne vysvetlenia zo SHAP

Pozorný čitateľ si mohol všimnúť, že napriek tvrdeniu v úvode, podľa ktorého metódy SHAP umožňujú generovať aj globálne vysvetlenia, sa doterajšia diskusia sústredila výlučne na lokálne vysvetlenia. Nejde o chybu. Metódy SHAP sú totiž primárne navrhnuté na vysvetľovanie individuálnych predikcií. Globálne vysvetlenia, je ale možné odvodiť agregáciou lokálnych SHAP hodnôt naprieč viacerými vzorkami [23]. Pre zachovanie parity s ostatnými použitými vysvetľovacími metódami, sa však v tejto práci budeme zameriavať výhradne na vysvetlenia lokálnych predikcií. Preto si tvorbu globálnych vysvetlení zo SHAP popíšeme len v krátkosti. Typicky sa globálne dôležitosti počítajú ako priemer absolútnych hodnôt jednotlivých prínosov pre každý príznak v rámci celého trénovacieho datasetu, formálne:

$$\mathcal{G}(\phi_i) = \frac{1}{n}\sum_{j=1}^{n} |\phi_{i,j}|$$

Kde n predstavuje počet vzoriek a $\phi_{i,j}$ označuje SHAP hodnotu príznaku i pre vzorku j. Táto agregácia kladie dôraz na celkovú dôležitosť jednotlivých príznakov. Alternatívne je možné počítať aj sumu alebo priemer pôvodných hodnôt (vrátane znamienka), čo umožňuje analyzovať aj smer vplyvu (pozitívny alebo negatívny) príznakov v rámci dát. Metódy SHAP disponujú tiež viacerými vizualizačnými technikami, pričom jednou z najčastejšie využívaných pre globálne vysvetlenia je *beeswarm* graf. Ten súčasne zobrazuje všetky lokálne hodnoty, čím poskytuje prehľad o priemernom vplyve a variabilite každého príznaku.

3.3 Anchors

Anchors je modelovo-agnostická metóda na lokálne vysvetľovanie predikcií, ktorú v článku [31] predstavili Ribeiro a kol. ako nástupcu LIME. Jej základom je tzv. *anchor*. Anchor je pravidlo typu IF-THEN vyjadrené v priestore príznakov, ktoré zaručuje stabilitu predikcie v okolí konkrétnej vzorky. Ak sú podmienky definované v pravidle

splnené, model s pravdepodobnosťou aspoň τ priradí rovnakú triedu ako pre pôvodnú vzorku. Pravidlá sú tiež doplnené o hodnotu presnosti (angl. *precision*) a odhadu pokrytia (angl. *coverage*). Metóda tak neponúka len samotné vysvetlenie predikcie, ale aj určitú mieru dôveryhodnosti tohto vysvetlenia. Štúdia vykonaná autormi v pôvodnom článku ukázala, že neodborní používatelia dokázali pomocou metódy Anchors predpovedať správanie modelu na nových vzorkách presnejšie a bez menšej námahy než s LIME alebo bez žiadneho vysvetlenia.

Metóda Anchors funguje tak, že hľadanie pravidla formuluje ako sekvenčný rozhodovací problém [23][31]. Pre danú vzorku x sa najprv definuje distribúcia perturbácií D_x, ktorá generuje syntetických susedov. Táto distribúcia je vytvorená upevnením príznakov v anchor kandidátovi na ich hodnoty pre vzorku x, pričom náhodne vzorkuje zvyšné príznaky. Kategorické príznaky sa obvykle vzorkujú na základe frekvencie v podkladovom datasete. Číselné príznaky môžu byť diskretizované alebo mierne upravené v rámci realistických hodnôt. Pri štruktúrovaných dátach, ako sú texty alebo obrázky, sú používané viac špecifické typy perturbácií, ako napr. náhodné odstraňovanie slov vo vete alebo maskovanie častí obrázka. Kľúčovou požiadavkou na všetky perturbácie však zostáva generovanie realistických variácií. Ak by nebola naplnená, z nich vytvorené odhady presnosti a pokrytia môžu byť skreslené alebo nespoľahlivé.

Algoritmus začína s prázdnym pravidlom a postupne pridáva predikáty[10], pričom každého kandidáta vyhodnocuje pomocou generovaných perturbácií. Počas tohto procesu sa priebežne odhaduje presnosť. Definovaná je ako podiel susedov, pri ktorých sa predikcia modelu zhoduje s pôvodnou predikciou $f(x)$. Keďže sa presnosť počíta priebežne počas vzorkovania, algoritmus môže včas posúdiť, ktoré pravidlá rozšíriť alebo naopak vyradiť. Algoritmus zároveň sleduje aj pokrytie anchor kandidátov, ktoré je definované ako podiel perturbovaných vzoriek spĺňajúcich podmienky pravidla. Vysoké pokrytie znamená, že predikcia modelu ostáva stabilná vo veľkej časti vstupného priestoru, kde platí dané pravidlo. Nízke pokrytie zas naznačuje síce úzko platné, ale stále spoľahlivé vysvetlenie.

Keďže množina možných pravidiel rastie exponenciálne, Anchors využíva kombináciu *beam search* a stratégiu Multi-Armed Bandit (MAB) na efektívne vyradenie pravidiel, ktoré pravdepodobne nedosiahnu požadovanú presnosť. Tento prístup minimalizuje počet potrebných volaní modelu tým, že včas odstraňuje nevhodných kandidátov. Vyhľadávanie sa zastaví, keď algoritmus nájde najmenšie pravidlo, ktorého dolná hranica presnosti prekračuje používateľom zadaný prah τ. V prípade viacerých akceptovateľných možností sa vyberie pravidlo s najväčším pokrytím. Celý postup je úplne post-hoc a nepredpokladá žiadnu znalosť vnútornej štruktúry modelu.

[10] obmedzenia typu príznak-hodnota

Výhodou metódy je formát IF-THEN pravidiel, keďže je zvyčajne ľahšie čitateľný a lepšie pochopiteľný než napr. číselné prínosy príznakov z metódy SHAP. Navyše, informácia o pokrytí upresňuje oblasť, pre akú je pravidlo platné, čím sa pomáha vyhnúť nesprávnej generalizácii na základe lokálnych poznatkov. Ak algoritmus nájde pravidlo s dostatočnou presnosťou, modelová predikcia zostáva stabilná aj pri zmenách ostatných príznakov. Opäť, v kontraste so SHAP, ktorý rozdeľuje zásluhy medzi všetky vstupné príznaky bez určenia, ktoré z nich rozhodujúcim spôsobom ovplyvňujú predikciu. Metóda Anchors však nie je bez obmedzení. Kvalita vysvetlení je priamo závislá od toho, ako realistické sú vzorkovacie distribúcie. Najmä pri obrázkoch a vysoko štruktúrovaných vstupoch nie je definovanie takéhoto rozdelenia triviálne a môže skresliť presnosť aj pokrytie. Taktiež ladenie hyperparametrov má výrazný vplyv na kvalitu výsledkov. Nevhodné nastavenia môžu viesť k pravidlám, ktoré sú natoľko špecifické (nízke pokrytie), že sú aplikovateľné len na niekoľko veľmi podobných vstupov. Naopak, príliš všeobecné pravidlá (nízka presnosť) sú síce aplikovateľné na veľké množstvo vzoriek, ale poskytujú málo informatívne vysvetlenia. Navyše, keďže presnosť sa odhaduje stochasticky, výsledky môžu byť medzi spusteniami odlišné, čím nie je zaručená reprodukovateľnosť. Možno najväčším záporom však zostáva výpočtová zložitosť algoritmu. Je síce nižšia než pri hľadaní riešenia hrubou silou, čas výpočtu však ostáva vysoký, najmä pri vysoko-dimenzionálnych vstupoch. Výpočtová náročnosť je totiž dominovaná členom úmerným p^2, kde p je počet príznakov. Dôvodom je mechanizmus MAB, ktorý v každom kroku vyberá najlepších kandidátov z veľkého množstva možností.

3.4 Kontrafaktuálne vysvetlenia

Kontrafaktuálne vysvetlenia (skr. kontrafaktuály) predstavujú metodologický prístup na vytváranie lokálnych vysvetlení modelových predikcií tým, že hľadajú minimálne zmeny hodnôt vstupných príznakov, ktoré by viedli k odlišnému výstupu modelu [23]. Tento prístup presúva pozornosť od abstraktného hodnotenia dôležitosti príznakov či analýzy vnútorného fungovania modelu k intuitívnym a prakticky využiteľným vysvetleniam. Obzvlášť je užitočný v situáciách, keď používateľ potrebuje zistiť, čo musí zmeniť, aby dosiahol požadovaný výsledok.

Tvorba kontrafaktuálov zvyčajne zahŕňa optimalizačné či vyhľadávacie techniky, ktorých cieľom je nájsť čo najmenšiu množinu zmien príznakov potrebných pre zmenu pôvodnej predikcie. Formálne, pre daný vstup x a výstup modelu $f(x)$ sa hľadá kontrafaktuál x' tak, aby platilo $f(x) \neq f(x')$ a zároveň bola vzdialenosť medzi x a x' čo najmenšia vzhľadom na zvolenú metriku. Výber metriky (napr. Euklidovská, Manhattanská alebo špecifická pre doménu) je kľúčový, pretože priamo ovplyvňuje realistickosť a praktickosť výsledných vysvetlení. Pri generovaní kontrafaktuálov sa často uplatňujú

aj praktické a etické obmedzenia, aby boli výsledky uskutočniteľné a spoločensky akceptovateľné. Napríklad vek môže len rásť a pohlavie si používateľ zvyčajne nemôže svojvoľne zmeniť.

Keďže ide o všeobecný prístup, nie konkrétnu metódu, existuje množstvo rôznych implementácií. Podľa Guidottiho článku [32] možno metódy generovania kontrafaktuálov rozdeliť do štyroch hlavných kategórií:

- **Optimalizačné**: Založené sú na definovaní stratovej funkcie, ktorá zachytáva požadované vlastnosti kontrafaktuálov, ako je napríklad blízkosť k pôvodnému vstupu, minimálny počet zmien či realistickosť. Na minimalizáciu tejto funkcie následne používajú optimalizačné algoritmy. Tieto metódy sú veľmi flexibilné a umožňujú zahŕňať rôzne obmedzenia, avšak sú často výpočtovo náročné. Klasickým reprezentantom je metóda od Wachter a kol. [33], ktorá využíva váženú Manhattanskú vzdialenosť, kde váhy sú určené inverznou mediánovou absolútnou odchýlkou jednotlivých príznakov. Príklad modernejšej metódy je Counterfactual Explanations Guided by Prototypes (CEGP) [34] od Van Looverena a Klaiseho, ktorá rozširuje stratovú funkciu o člen, ktorý smeruje kontrafaktuál k prototypu cieľovej triedy. Prototyp je definovaný ako priemer k najbližších susedov v latentnom priestore autoenkódera.

- **Heuristické vyhľadávanie**: Spoliehajú sa na heuristické, často pažravé algoritmy, ktoré iteratívne prehľadávajú priestor možných vstupov a v každom kroku vykonávajú lokálne rozhodnutia na minimalizáciu cenovej funkcie. Hoci sú výpočtovo efektívnejšie než optimalizačné prístupy, neposkytujú vždy optimálne výsledky. Príkladom je metóda Growing Spheres Generation (GSG) [35] od Laugel a kol., ktorá generuje syntetické vzorky v priestore príznakov v L2-guľovej vrstve okolo vstupu x a zmenšuje polomer generovania, až kým nenájde platný kontrafaktuál.

- **Založené na inštanciách**: Predstavujú viac dátovo orientovaný prístup. Kontrafaktuály hľadajú priamo v podkladovom datasete, pričom vyberá vzorky, ktoré sú podobné pôvodnému vstupu, ale vedú k inému rozhodnutiu modelu. Týmto spôsobom je zabezpečená realistickosť kontrafaktuálov, keďže vychádzajú zo skutočných dát. Ako príklad možno uviesť metódu Feasible and Actionable Counterfactual Explanations (FACE) [36] od Poyiadzi a kol., ktorá vytvára graf z dátových vzoriek a hľadá najkratšiu cestu, s ohľadom na hustotu, vedúcu ku kontrafaktuálu.

- **Založené na rozhodovacích stromoch**: Odhadujú správanie modelu čiernej skrinky pomocou náhradného rozhodovacieho stromu. Po natrénovaní náhradného modelu je možné generovať kontrafaktuály prechádzaním stromovej štruktúry, kde

sú identifikované minimálne zmeny príznakov vedúcich k inému výstupu. Tento prístup je prirodzene interpretovateľný vďaka jednoduchej logike rozhodovacích ciest. Príkladom je metóda Local Rule-based Explainer (LORE) [37] od Guidottiho a kol., ktorá generuje syntetických susedov okolo vstupu x a trénuje na nich rozhodovací strom. Z rozhodovacej cesty potom vytvára sadu pravidiel vrátane kontrafaktuálnych scenárov, ktoré ukazujú alternatívne rozhodnutia.

Kontrafaktuálne vysvetlenia majú mnoho výhod, ktoré prispeli k ich širokej adopcii, hlavnou je ale určite ich intuitívnosť [23]. Sú ľahko pochopiteľné aj pre neodborníkov, keďže priamo navrhujú kroky, ktoré môže používateľ vykonať pre dosiahnutie požadovaného výsledku. Vzhľadom na svoju formu „čo ak“ scenárov sa tiež prirodzene zhodujú so spôsobom, akým ľudia uvažujú o kauzalite. Tým podporujú lepšie pochopenie správania modelu a zvyšujú dôveru v jeho rozhodnutia, aj v oblastiach s vysokými nárokmi na transparentnosť. Zároveň však čelia viacerým výzvam. Generovanie zmysluplných a minimálnych kontrafaktuálov môže byť výpočtovo náročné, obzvlášť pri komplexných nelineárnych modeloch alebo vysoko-dimenzionálnych dátach. Okrem toho, niektoré kontrafaktuály môžu navrhovať nerealistické alebo eticky problematické zmeny, ako je úprava veku, pohlavia či etnickej príslušnosti a podobne. Aj takéto nevhodné návrhy však môžu byť užitočné pri odhaľovaní zaujatostí modelu (napr. ak schválenie hypotéky závisí výlučne od zmeny pohlavia). Problémom môže byť aj existencia viacerých platných kontrafaktuálnych vysvetlení pre jednu vstupnú vzorku, občas dokonca zdanlivo navzájom odporujúcich. To môže byť pre používateľa mätúce a predstavuje výzvu pri rozhodovaní, ktoré z vysvetlení označiť ako to „najlepšie“.

4 Zber a predspracovanie dát

V tejto kapitole sa podrobne pozrieme na celý proces prípravy dát, od ich výberu až po úpravu do formátu vhodného na trénovanie klasifikačných modelov. V podkapitole 4.1 začneme analýzou požiadaviek na vhodný dataset a predstavíme niekoľko verejne dostupných datasetov, ktoré sme zvažovali. Porovnáme ich silné a slabé stránky a vysvetlíme, aké kritériá rozhodli o našom výbere. V nasledujúcej časti 4.2 potom podrobne opíšeme všetky kroky predspracovania, ktoré sme vykonali na vytvorenie finálneho datasetu pripraveného na použitie pri trénovaní spoľahlivých modelov.

4.1 Požiadavky a prehľad datasetov

Kvalitné dáta sú základom úspešných modelov strojového učenia. Inak tomu nie je ani pri detekcii kategórií malvéru. Pri výbere vhodného datasetu sme preto definovali niekoľko kľúčových požiadaviek, ktoré musí spĺňať, aby bolo možné natrénovať spoľahlivé a výkonné modely:

1. **Dynamické vlastnosti**: Dataset musí obsahovať dáta získané prostredníctvom dynamickej analýzy správania malvéru. Typickými príkladmi sú API volania, systémové volania, prístupy k súborom a registru, sieťová komunikácia či procesová aktivita. Tento typ dát poskytuje komplexný pohľad na činnosť malvéru v systéme, čo je ideálne pre trénovanie modelov. Detailnejší opis dynamickej analýzy sa nachádza v časti 1.3.1.

2. **Viaceré kategórií**: Dataset musí obsahovať vzorky reprezentujúce rôzne typy malvéru. Ich prítomnosť je nevyhnutná pre natrénovanie modelov, ktoré dokážu spoľahlivo rozlišovať medzi viacerými kategóriami malvéru. Kategórie malvéru sú podrobne rozobrané v časti 1.2.

3. **Označené vzorky**: Každá vzorka v datasete musí byť jednoznačne označená konkrétnou kategóriou malvéru. Bez jasného označenia by nebolo možné modely správne trénovať ani hodnotiť ich výslednú presnosť.

4. **Dostatočný počet vzoriek**: Každá kategória musí byť zastúpená dostatočným počtom reprezentatívnych vzoriek. Príliš malý počet vzoriek by mohol viesť k zlej generalizácii modelu pri nových, pred tým nevidených dátach. Ideálne by bolo, ak by počty vzoriek medzi kategóriami boli vyvážené, alebo aspoň nie výrazne nevyvážené.

5. **Dostupnosť pre akademické účely**: Dataset musí byť legálne dostupný pre výskumné a akademické použitie. Preferujeme open-source datasety alebo také, ktoré majú jasne definovanú licenciu umožňujúcu akademické použitie.

Na základe vyššie uvedených požiadaviek sme identifikovali niekoľko datasetov, ktoré sú potenciálne vhodné pre naše účely:

- **Windows PE Malware API dataset**: Malý dataset publikovaný ako súčasť výskumu [38] a voľne dostupný cez platformu GitHub [39]. Obsahuje dynamicky získané dáta o správaní 582 malvérových a 439 legitímnych Windows PE súborov. Konkrétne ide o frekvencie Windows API volaní zaznamenané počas analýzy v prostredí *Cuckoo Sandbox*. Malvérové vzorky sú rozdelené do 14 kategórií na základe analýzy pomocou VirusTotal API. Dataset však neobsahuje informácie o poradí vykonaných API volaní, iba ich celkové počty. V súvislosti s týmto obmedzením sme kontaktovali jednu z autoriek datasetu, no do uzávierky práce sme nedostali odpoveď.

- **New Datasets for Dynamic Malware Classification**: Kolekcia dvoch stredne veľkých datasetov predstavených v práci [40]. Oba datasety sú verejne dostupné pre akademické účely cez GitHub [41]. Prvý dataset *VirusSample* obsahuje 9 795 vzoriek malvéru naprieč 13 rôznymi kategóriami. Druhý, väčší dataset *VirusShare* obsahuje až 14 616 vzoriek a 15 odlišných kategórií. Datasety obsahujú sekvencie volaní spolu s hašmi jednotlivých vzoriek a ich klasifikáciou podľa VirusTotal. Sekvencie volaní zahŕňajú prevážne Windows API, ale tiež funkcie z C/C++ runtime knižníc a iných bežných systémových knižníc. Oba datasety sú však značne nevyvážené a dominované vzorkami pre trojan, ktoré predstavujú približne 60% všetkých dát. Za nimi nasledujú vzorky pre vírus s podielom 24% pre VirusSample a 17% pre VirusShare. Zvyšné kategórie sú zastúpené v oveľa menšej miere.

- **Mal-API-2019**: Stredne malý dataset publikovaný v štúdii [42] a prístupný cez GitHub [43]. Zahŕňa 7 107 vzoriek malvéru rozdelených do 8 kategórií. Zložený je zo sekvencií Windows API volaní, ktoré boli zaznamenané pomocou *Cuckoo Sandbox* a kategorizácia bola opäť realizovaná prostredníctvom VirusTotal. Dataset je pomerne dobre vyvážený. Väčšina kategórií obsahuje približne 1 000 vzoriek, s výnimkou kategórie advér (379 vzoriek).

Ani jeden z analyzovaných datasetov však nespĺňa všetky stanovené požiadavky. Prvý dataset obsahuje málo malvérových vzoriek, ktoré sú navyše ešte rozdelené medzi príliš veľké množstvo kategórií. Pri podrobnejšej analýze sme navyše zistili, že viaceré

vzorky majú zaznamenané nulové frekvencie API volaní alebo im chýbajú označenia kategórie. Takéto dáta sú pre účely trénovania nepoužiteľné a ich odstránenie by ešte viac znížilo už tak obmedzený počet relevantných vzoriek. Modely natrénované na takto malom počte dát by nedokázali generalizovať na nové dáta. Navyše, chýbajúce informácie o poradí volaní znemožňujú aplikáciu metód založených na sekvenčných charakteristikách, ako sú napríklad n-gramy.

Kolekcia dvoch datasetov VirusSample a VirusShare disponuje dostačujúcim počtom vzoriek a má aj dobré pokrytie rôznych kategórií, trpí však výraznou nevyváženosťou. Takéto rozdelenie môže viesť k tomu, že model bude výrazne preferovať dominantné triedy, zatiaľ čo menej zastúpené kategórie bude klasifikovať s nižšou presnosťou alebo ich úplne ignorovať. Výhodou týchto datasetov je však prítomnosť sekvencií volaní, ktoré umožňujú využitie pokročilejších metód na zachytenie väčšieho kontextu správania. K lepšiemu pochopeniu správania modelu prispieva aj rôznorodosť typov príznakov, keďže sú prítomné volania od Windows API až po funkcie z runtime a systémových knižníc. Toto má potenciál zvýšiť presnosť klasifikácie modelov, keďže umožňujú modelu rozpoznávať kategórie malvéru aj na základe menších rozdielov v interakcii so systémovým prostredím. Na druhej strane však táto variabilita výrazne zvyšuje dimenzionalitu dát, čo vedie ku komplexnejším modelom. Tie sú náročnejšie na trénovanie aj na aplikáciu vysvetľovacích metód. Pôvodne sme predpokladali, že ide o dva samostatné datasety s odlišnými vzorkami. Túto skutočnosť podporoval aj publikovaný článok [40], podľa ktorého boli oba datasety získané z rôznych zdrojov. Počas detailnejšej analýzy sme však zistili, že všetky haše vzoriek z datasetu VirusSample sa nachádzajú aj v datasete VirusShare. Z toho vyplýva, že nejde o dva samostatné datasety, ale že VirusSample predstavuje podmnožinu datasetu VirusShare.

Posledný dataset Mal-API-2019 je veľmi dobre vyvážený, keďže väčšina tried, až na jednu výnimku, je zastúpená v približne rovnakom počte. Vďaka tomu má natrénovaný model potenciál rovnako dobre rozpoznávať rôzne kategórie malvéru. Počet zastúpených kategórií je taktiež dostačujúci, nevýhodou je však menší počet vzoriek, ktorý môže negatívne ovplyvniť generalizáciu modelu. Pozitívom je však prítomnosť sekvenčných volaní, ktoré umožňujú využiť väčší kontext správania malvéru na klasifikáciu.

Na základe uvedenej analýzy sme sa rozhodli neobmedziť iba na jeden konkrétny dataset, ale skombinovať dáta z VirusShare a Mal-API-2019. Takýmto spojením získame rozsiahlejší a rozmanitejší dataset, ktorý obsahuje dostatočný počet vzoriek, široké spektrum kategórií a zároveň zachováva sekvenčnú štruktúru dát. Prvý dataset sme sa rozhodli nezaradiť najmä kvôli chýbajúcim informáciám o poradí volaní. Jeho pridaním by sme totiž prišli o možnosť využiť sekvenčné charakteristiky dát, ktoré sú pri detekcii malvéru často kľúčové. Uvedomujeme si, že novo vzniknutý dataset bude trpieť

nevyváženosťou medzi jednotlivými kategóriami, no vnímame to ako menej závažné než nedostatočný počet vzoriek. Navyše existujú známe prístupy na zmiernenie tohto problému, ako napríklad nadvzorkovanie (angl. *oversampling*) menšinových tried alebo použitie váhovania pri trénovaní modelov.

4.2 Predspracovanie dát

Pred samotným trénovaním modelov je nevyhnutné, aby dáta prešli predspracovaním. Táto fáza je obzvlášť dôležitá vzhľadom na rozhodnutie kombinovať dáta z dvoch rôznych datasetov, ktoré sa líšia vo formáte, rozsahu aj kvalite dát.

Cieľom predspracovania je vytvoriť jednotný a konzistentný formát, ktorý bude kompatibilný s požadovanými krokmi trénovania. Výsledný dataset bol preto navrhnutý vo formáte CSV so štruktúrou troch stĺpcov: `api`, `class` a `api_count`. Stĺpec `api` bude obsahovať sekvencie API volaní[11] oddelené čiarkami, čo umožní jednoduché spracovanie pomocou techník ako `CountVectorizer`. Stĺpec `class` bude reprezentovať cieľovú kategóriu malvéru a `api_count` udávať celkový počet API volaní pre danú vzorku.

Dataset VirusShare bol už v takmer požadovanom formáte a preto si vyžadoval len minimálne úpravy. Sekvencie API volaní a príslušné kategórie boli priamo dostupné a bolo ich možné jednoducho zaradiť do výsledného formátu. Naopak, dataset Mal--API-2019 si vyžadoval rozsiahlejšie úpravy. Dáta boli pôvodne rozdelené do dvoch súborov. API volania sa nachádzali v textovom súbore a kategórie v CSV súbore. Samotné volania v sekvenciách boli oddelené medzerami, ktoré sme nahradili čiarkami, aby sme zabezpečili jednotný formát. Následne sme sekvencie spárovali s odpovedajúcimi kategóriami a vytvorili z nich spoločnú tabuľku. Po vykonaní týchto úprav boli dáta z oboch datasetov zlúčené do jedného celku. Počet volaní pre vzorky doplníme v neskoršej fáze. V datasete sa v tomto bode nachádzalo 21 722 rozličných vzoriek a 23 596 unikátnych API volaní. Celkové rozdelenie vzoriek podľa tried sa nachádza v tabuľke číslo 1.

Na prvý pohľad je z distribúcie zrejmé, že počet vzoriek sa medzi jednotlivými kategóriami výrazne líši. Okrem celkovej nevyváženosti, triedy *agent*, *ransomvér* a zvyšné ešte menej početné kategórie sú v datasete zastúpené len v minimálnom množstve. Vzhľadom na extrémne nízky počet vzoriek by pre tieto triedy nebolo možné vytvoriť dostatočne reprezentatívne trénovacie, validačné ani testovacie množiny. V tomto prípade by nepomohli ani techniky nadvzorkovania, pretože by viedli len k duplikácii niekoľkých

[11] Od tohto bodu budeme pre zjednodušenie pod pojmom API volania rozumieť všetky typy dynamicky získaných volaní, vrátane Windows API, funkcií z runtime knižníc, systémových knižníc a podobne. V prípadoch, kde bude potrebné rozlišovať konkrétny typ volania, to výslovne uvedieme.

Tabuľka 1: Distribúcia tried pred predspracovaním

Kategória	**Počet vzoriek**
Trojan	9 920
Vírus	3 491
Červ	1 525
Backdoor	1 511
Advér	1 287
Downloader	1 219
Dropper	931
Spajvér	877
Nedefinované	576
Agent	165
Ransomvér	115
Riskvér	85
Crypt	10
Keylogger	7
Rootkit	3
Spolu	**21 722**

málo dostupných vzoriek. To by viedlo k preučeniu modelu a neschopnosti generalizovať na nové dáta. Z týchto dôvodov sme sa rozhodli odstrániť z datasetu vzorky patriace do tried *agent*, *ransomvér*, *riskvér*, *crypt*, *keylogger* a *rootkit*. Pozornosť si však vyžaduje aj trieda označená ako *nedefinované*. Pochádza výlučne z datasetu VirusShare, no v článku [40] sa jej význam ani pôvod neuvádza. Predpokladáme však, že ide o vzorky malvéru, ktoré sa autorom nepodarilo zaradiť ani do jednej z tried. Naším cieľom je však klasifikácia do známych a jasne definovaných kategórií malvéru, rozhodli sme sa preto takto označené vzorky z datasetu taktiež odstrániť. Po odstránení tried s malým výskytom a triedu *nedefinované*, nám v datasete zostalo 20 761 vzoriek.

Pri detailnejšej analýze sekvencií volaní sme vo viacerých vzorkách objavili tzv. *mangled* názvy. Ide o zakódované symboly, typické pre jazyk C++, ktoré vznikajú počas kompilácie v procese známom ako *name mangling* [44]. Tento mechanizmus vkladá do názvov funkcií a premenných dodatočné informácie, ako sú typy parametrov, návratové hodnoty, názvy tried či menné priestory. Vďaka tomu dokáže kompilátor rozlíšiť medzi entitami s rovnakým identifikátorom, napríklad pri preťažovaní funkcií alebo v prípade rovnomenných prvkov v rôznych rozsahoch viditeľnosti. Výsledkom sú

špecificky upravené názvy symbolov, ktoré linker používa na ich správne rozpoznanie a spojenie počas zostavovania programu. Takto zakódované názvy je možné rozpoznať podľa ich neobvyklého formátu. V prípade jazyka C++ sa často začínajú znakmi `?` (pre MSVC) alebo `_Z` (pre GCC alebo Clang) Tu je niekoľko príkladov mangled názvov, ktoré sa objavili v našom datasete:

```
?GetProcAddress@egl@@YGP6AXXZPBD@Z
?SwapBuffers@egl@@YGIPAX0@Z
_Z11qUncompressPKhi
??0FLAG_ARGUMENT@@QAE@XZ
_ZNK17QNetworkInterface15hardwareAddressEv
```

Na ich dekódovanie je potrebné použiť reverzný proces, tzv. *demangling*. K dispozícii je viacero voľne dostupných nástrojov, ktoré tento proces umožňujú. Uviesť je možné napríklad `undname` pre Windows, `c++filt` na Unixových systémoch, alebo tiež webové nástroje ako je `demangler.com`. Posledný spomenutý bol použitý na získanie demangled výstupu vyššie uvedených príkladov:

```
void (__cdecl*__stdcall egl::GetProcAddress(char const *))(void)
unsigned int __stdcall egl::SwapBuffers(void *,void *)
qUncompress(unsigned char const*, int)
public: __thiscall FLAG_ARGUMENT::FLAG_ARGUMENT(void)
QNetworkInterface::hardwareAddress() const
```

Z hľadiska modelov na detekciu malvéru však tieto hodnoty nie sú praktické. Ich prítomnosť ešte nezaručuje, že daná funkcia bola skutočne zavolaná, keďže ide len o deklarácie alebo exportované symboly. Navyše, tieto informácie nemusia byť spoľahlivé. Nachádzajú sa totiž zvyčajne v exportovacích tabuľkách, debug informáciách či tabuľkách symbolov binárneho súboru, ktoré tvorcovia malvéru dokážu modifikovať alebo úplne odstrániť. Problémom je tiež ich závislosť od konkrétneho kompilátora či platformy, keďže formát manglingu nie je štandardizovaný. Okrem toho sa v našom datasete vyskytujú veľmi zriedkavo, čo by pri ich ponechaní viedlo len k zbytočnému zvýšeniu dimenzionality bez výrazného prínosu k presnosti modelov. Z týchto dôvodov sme sa rozhodli mangled hodnoty z datasetu odstrániť. Detekcia prebiehala iteráciou nad sekvenciou volaní pre každú vzorku, pričom ak sme narazili na hodnotu, ktorá sa začínala na jeden z poznávacích znakov, tak túto hodnotu sme zo sekvencie vyradili. Celkovo sme z datasetu vyradili až 13 404 mangled hodnôt.

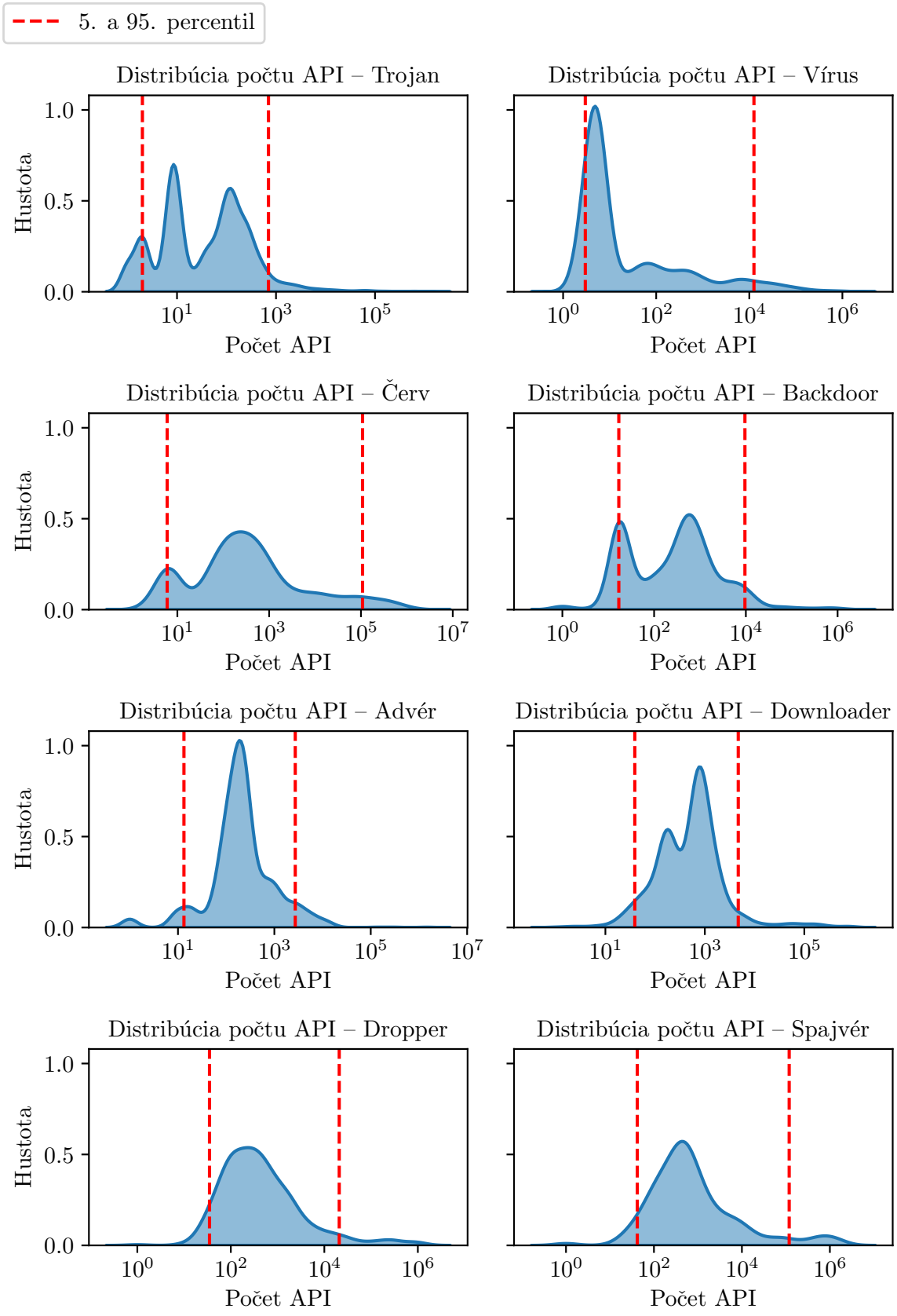
Následne sme vykonali kontrolu nesprávnych a chýbajúcich hodnôt v stĺpci `api`. Tie sa mohli v datasete vyskytovať už od začiatku, alebo vzniknúť po odstránení takého

veľkého množstva mangled volaní. Žiadne takéto hodnoty sme však nezaznamenali, počet vzoriek zostal preto nezmenený a teda 20 761.

Posledným krokom v predspracovaní bola analýza počtu API volaní v jednotlivých vzorkách, ktorá nám poslúžila na identifikáciu hraničných hodnôt. Ako sme načrtli na začiatku, do datasetu sme doplnili stĺpec `api_count`, ktorý obsahuje počet volaní v sekvencii zo stĺpca `api`. Následne sa nám podarilo určiť, že počet volaní sa medzi vzorkami pohybuje od 1 až po 1 764 421. Jedná sa síce o extrémne široký rozsah, to však neznamená, že je nevyhnutne problémový. Rôzne kategórie malvéru môžu totiž prirodzene generovať rôzne dlhé sekvencie volaní. Priame odstránenie takýchto globálnych hraničných hodnôt by potom mohlo viesť k strate dôležitých informácií. Vhodnejšie bolo preto vykonať analýzu počtu volaní podľa tried. Na tento účel sme si zvolili graf typu jadrový odhad hustoty (angl. *Kernel Density Estimation (KDE)*). Ten nám umožní vizualizovať pravdepodobnostné rozdelenia počtu volaní a vďaka tomu presnejšie identifikovať hraničné hodnoty. Vzhľadom na výrazné rozdiely v rozsahu sme na osi x použili logaritmickú mierku. Výsledný graf sa nachádza na obrázku číslo 1.

Z grafu jasne vyplýva, že rôzne triedy malvéru disponujú odlišnými distribučnými vzormi v počte API volaní. Napríklad kategória trojan má dva hlavné vrcholy. Prvý v rozmedzí desiatok a druhý v stovkách volaní, pričom výskyty nad tisíc volaní sú už len veľmi zriedkavé. V prípade kategórie vírus pozorujeme výrazný vrchol v oblasti desiatok volaní, no pri vyšších hodnotách dochádza k prudkému poklesu. Trieda červ má najväčšiu hustotu v rozmedzí stoviek až tisícok API volaní, no menší vrchol sa vyskytuje aj v desiatkach. Podobne ako trojan, má trieda backdoor dva vrcholy. Prvý v desiatkach, no druhý opäť až v tisíckach volaní. Pri kategóriách advér, downloader, dropper a spajvér pozorujeme dominantný výskyt v rozmedzí stoviek až tisícok API volaní. Nižšie aj vyššie hodnoty sa vyskytujú len zriedka. Tieto rozdiely potvrdzujú náš predpoklad, že počet API volaní nie je rovnomerne rozložený naprieč triedami, ale je do značnej miery ovplyvnený správaním konkrétneho typu malvéru. Z tohto dôvodu by odstránenie globálnych hraničných hodnôt nebolo dobrým riešením. Ako lepší prístup sa však ukázalo použitie percentilového rezu, konkrétne 5. a 95. percentilu. Tieto hranice sú už v grafe vyznačené prerušovanými červenými čiarami. Ako vidno z grafu, takýto rez nám umožní zachovať väčšinu relevantných dát, pričom sa zároveň zbavíme hraničných hodnôt, ktoré by mohli negatívne ovplyvniť kvalitu modelov. Celkovo sme odstránili 1 834 vzoriek.

Týmto je predspracovanie hotové a dataset je pripravený na ďalšie použitie v procese trénovania modelov. Vo výslednom datasete nám zostalo 18 927 vzoriek a 10 513 unikátnych API volaní. Tabuľka 2 zobrazuje finálnu distribúciu vzoriek podľa tried a zároveň uvádza hranice 5. a 95. percentilu počtu API volaní, ktoré boli použité



Obr. 1: KDE distribúcia počtu API volaní podľa tried

Tabuľka 2: Distribúcia tried po predspracovaní, doplnená o dolné a horné hranice počtu API volaní podľa 5. a 95. percentilu

Kategória	**Počet vzoriek**	**Dolná hranica**	**Horná hranica**
Trojan	8 938	2	705
Vírus	3 276	3	12 592
Červ	1 412	6	108 355
Backdoor	1 410	17	9 542
Advér	1 161	13	2 714
Downloader	1 099	39	4 699
Dropper	839	35	20 820
Spajvér	792	41	118 814
Spolu	**18 927**		

na odstránenie hraničných hodnôt. Hoci trieda trojan zostáva s veľkým náskokom najpočetnejšia, očakávame, že aj ostatné kategórie budú obsahovať dostatočný počet reprezentatívnych vzoriek na to, aby umožnili natrénovať modely schopné generalizovať aj na nové, doposiaľ nevidené dáta.

5 Implementácia klasifikátorov

V tejto kapitole sa zameriame na implementáciu troch klasifikačných modelov, ktoré sme použili na kategorizáciu malvéru na základe predspracovaných dát, opísaných v predchádzajúcej časti. Zvolili sme si algoritmy náhodný les, XGBoost a viacvrstvový perceptrón, pričom každý z nich reprezentuje odlišný prístup ku klasifikácii. Ich použitie nám tak umožňuje porovnať správanie a výkonnosť rôznych prístupov v kontexte našej úlohy.

Náhodný les sme si zvolili ako klasického reprezentanta stromových metód. Je známy svojou odolnosťou voči šumu, nenáročným ladením a schopnosťou predchádzať preučeniu. Podrobnejšie sme ho opísali v podkapitole 2.1.

XGBoost bol vybraný ako moderný, výkonný algoritmus založený na princípe gradientového boostingu. Je známy tým, že dosahuje veľmi dobré výsledky v mnohých praktických úlohách s tabuľkovými dátami a ponúka široké možnosti ladenia parametrov. Jeho použitím sme chceli overiť, do akej miery sa v našom probléme potvrdí jeho výkonnostná prevaha. Detailnejší popis tohto algoritmu sa nachádza v podkapitole 2.2.

Viacvrstvový perceptrón sme si vybrali ako zástupcu prístupov založených na neurónových sieťach. Chceli sme overiť, ako si tento model poradí v porovnaní so stromovými algoritmami, najmä pokiaľ ide o schopnosť zachytiť komplexné nelineárne vzťahy v dátach. Bližšie sme si jeho fungovanie opísali v podkapitole 2.3.

Tréning všetkých troch modelov prebiehal na počítači s procesorom Intel Core i5-9300H (4 jadrá), GPU NVIDIA GeForce GTX 1650, pamäťou RAM s kapacitou 24 GB a operačným systémom Windows 11. Vývoj aj samotný tréning bol však realizovaný prostredníctvom WSL 2 so systémom Ubuntu 22.04.5.

Ako cieľ pre naše klasifikačné modely sme si určili prekonať výsledky uvedené v štúdii [40]. V nej autori taktiež implementovali modely RF a XGBoost, pričom sa im podarilo, pre nevyváženú verziu datasetu VirusShare, dosiahnuť pre model RF F1-skóre 0,602 a pre model XGBoost 0,718.

Túto kapitolu začneme opisom metód, ktoré sme použili na overovanie a vyhodnocovanie výsledkov klasifikátorov (podkapitola 5.1). Pokračovať budeme popisom transformácie dát, nevyhnutnej pre spracovanie sekvenčných textových dát do podoby vhodnej na trénovanie (podkapitola 5.2). Osobitnú pozornosť sme tiež venovali procesu ladenia hyperparametrov vektorizácie, ktorý mal zásadný vplyv na výkon modelov (5.2.1). Následne sa zameriame na jednotlivé klasifikátory, pričom popíšeme použitú architektúru, proces ladenia hyperparametrov a dosiahnuté výsledky (podkapitoly 5.3, 5.4 a 5.5). Kapitolu zakončíme porovnaním a celkovým vyhodnotením výsledných

modelov (podkapitola 5.6) a detailnou analýzou výsledkov podľa jednotlivých kategórií (podkapitola 5.6.1).

5.1 Metódy overovania a hodnotenia výsledkov

Pre zabezpečenie spoľahlivého hodnotenia výkonnosti klasifikačných modelov sme v tejto práci aplikovali systematické metódy validácie a štandardné vyhodnocovacie metriky. Opis týchto prístupov uvádzame v nasledujúcich častiach.

5.1.1 Krížová validácia

Krížová validácia (angl. *cross-validation*) je zastrešujúci pojem pre štatistické metódy určené na hodnotenie generalizačnej schopnosti modelov [45]. Jej hlavným cieľom je minimalizovať riziko preučenia a zabezpečiť, že výkonnosť modelu nebude závisieť len od konkrétneho rozdelenia dát na trénovaciu a testovaciu množinu.

V tejto práci používame metódu *k-fold* krížovej validácie. Tá spočíva v rozdelení dát do k približne rovnako veľkých častí, pričom jedna takáto časť sa označuje ako *fold*. Model je následne natrénovaný k-krát, pričom v každej iterácii sa použije $k - 1$ foldov na trénovanie a zostávajúci fold na validáciu. Tento proces sa opakuje dovtedy, kým každý fold nebol použitý na validáciu práve raz. Vyhodnocovanie metriky sú potom zo všetkých iterácií spriemerované pre dosiahnutie výsledného odhadu výkonnosti modelu.

Výber hodnoty k závisí od požadovaného pomeru medzi výpočtovou náročnosťou a spoľahlivosťou hodnotenia. Nižšie hodnoty, ako $k = 3$, majú síce nižšie výpočtové nároky, no ich odhad výkonu môže byť zavádzajúci. Naopak, vyššie hodnoty ako $k = 10$ vedú k stabilným a nezaujatým odhadom, ale za cenu značnej výpočtovej záťaže. Počas ladenia modelov v tejto práci sme primárne využívali hodnotu $k = 5$, čiže 5-násobnú krížovú validáciu, keďže predstavuje dobrý kompromis medzi záťažou a spoľahlivosťou. Vo viacerých prípadoch sme však použili aj 10-násobnú krížovú validáciu, najmä keď sme potrebovali presnejšie výsledky. Pri popise výsledkov sme vždy uviedli, aká krížová validácia bola použitá.

Vzhľadom na veľmi nevyvážené zastúpenie kategórií malvéru v našom datasete bola použitá *stratifikovaná* verzia k-násobnej krížovej validácie. Tento variant zabezpečuje, že distribúcia tried je zachovaná v každom folde. Nemôže sa teda stať, že by sa v niektorom folde nenachádzala žiadna z minoritných vzoriek, čím sa predchádza skresleniu výsledkov.

5.1.2 Vyhodnocovacie metriky

Na posúdenie výkonnosti našich klasifikačných modelov sme použili množinu štandardne používaných metrík, a to správnosť, presnosť, senzitivitu a primárne F1-skóre. Pre

ich pochopenie je však najprv nutné objasniť základné pojmy, z ktorých tieto metriky vychádzajú. Pre jednoduchosť, v kontexte binárnej klasifikácie rozlišujeme [46]:

- **Skutočný pozitív** (angl. *True Positive*, skr. TP): Model správne označí pozitívnu vzorku (napr. súbor je malvér a model ho klasifikuje ako malvér).
- **Skutočný negatív** (angl. *True Negative*, skr. TN): Model správne označí negatívnu vzorku (napr. súbor je legitímny a model ho klasifikuje ako legitímny).
- **Falošný pozitív** (angl. *False Positive*, skr. FP): Model nesprávne označí negatívnu vzorku ako pozitívnu (napr. súbor je legitímny, ale model ho klasifikuje ako malvér).
- **Falošný negatív** (angl. *False Negative*, skr. FN): Model nesprávne označí pozitívnu vzorku ako negatívnu (napr. súbor je malvér, ale model ho klasifikuje ako legitímny).

V prípade viactriednej klasifikácie platia rovnaké koncepty, ale s tým rozdielom, že sa vyhodnocujú štýlom *per-class* použitím prístupu *one-vs-all*. To znamená, že pre každú triedu sa samostatne posudzuje výkonnosť modelu, ako keby daná trieda bola pozitívna a všetky ostatné triedy boli negatívne. Takto vypočítané metriky sa môžu následne agregovať (napr. pomocou makro priemeru). Na základe týchto konceptov sú potom odvodené nasledujúce metriky [47]:

- **Správnosť** (angl. *accuracy*): Vyjadruje podiel správne klasifikovaných vzoriek vzhľadom na celkový počet vzoriek. Formálna definícia:
$$\text{Správnosť} = \frac{TP + TN}{TP + TN + FP + FN}$$
Ide o jednoduchú a intuitívnu metriku, ktorá má potenciál poskytnúť dobrý odhad výkonu modelu. Nutnou podmienkou je však vyváženosť dát, v opačnom prípade je často zavádzajúca. Uprednostňuje totiž viacpočetné triedy. Aj napriek tomu je však zvykom ju uvádzať.
- **Presnosť** (angl. *precision*): Vyjadruje podiel správne klasifikovaných pozitívnych vzoriek spomedzi všetkých vzoriek, ktoré model označil ako pozitívne. Formálna definícia:
$$\text{Presnosť} = \frac{TP}{TP + FP}$$
Čím je presnosť vyššia, tým menej model vytvára falošne pozitívnych predikcií.
- **Senzitivita** (angl. *recall*): Určuje, aký podiel skutočne pozitívnych vzoriek dokázal model správne identifikovať. Formálna definícia:
$$\text{Senzitivita} = \frac{TP}{TP + FN}$$

Vyššia senzitivita je potrebná v prípadoch, keď je dôležité zachytiť čo najviac relevantných vzoriek, často aj za cenu vyššej chybovosti.

- **F1-skóre** (angl. *F1-score*, skr. F1): Harmonický priemer presnosti a senzitivity, vďaka čomu predstavuje akýsi kompromis medzi obomi metrikami. Formálna definícia:
$$\text{F1} = 2 \cdot \frac{\text{Presnosť} \cdot \text{Senzitivita}}{\text{Presnosť} + \text{Senzitivita}}$$
F1-skóre je obzvlášť užitočné pri nevyvážených datasetoch (ako je ten náš), keďže berie do úvahy ako falošne pozitívne, tak aj falošne negatívne predikcie. Z tohto dôvodu sme túto metriku používali ako primárny ukazovateľ pri hodnotení výsledkov klasifikátorov.

Na hodnotenie modelov budeme okrem priemerných hodnôt jednotlivých metrík uvádzať aj ich štandardnú odchýlku, ktorá vyjadruje variabilitu výsledkov medzi jednotlivými foldmi krížovej validácie. Výsledky budeme prezentovať vo formáte *priemer* ± *štandardná odchýlka*.

5.2 Transformácia dát

V tejto časti popisujeme proces transformácie dát, ktorý predstavuje východiskový bod pre celý tréningový proces. Napriek tomu, že naše dáta už prešli predspracovaním (podkapitola č. 4.2), v tejto forme ešte neboli vhodné na priamy vstup do modelov. Naším cieľom bolo teda previesť textové reťazce API volaní zo stĺpca `api` do číselného formátu, ktorý budú môcť modely použiť ako príznaky. Tento prevod textu do číselnej formy sa označuje ako vektorizácia.

Vzhľadom na štruktúru našich dát sme na túto konverziu zvolili metódu známu ako *bag-of-words*. Táto metóda prevádza každý dokument[12] na vektor zachytávajúci počty výskytov jednotlivých tokenov[13]. Vznikla tak riedka matica príznakov, kde každý stĺpec predstavuje určitý token (napr. konkrétne API volanie) a každé číslo v matici udáva jeho výskyt v danom dokumente.

Na implementáciu sme použili triedu `CountVectorizer` z knižnice `scikit-learn`. Táto trieda zabezpečuje nielen uvedené rozdelenie textu na tokeny, ale aj odstránenie špeciálnych znakov, prevod na malé písmená a ďalšie úpravy. Navyše podporuje aj tvorbu n-gramov. Ako sme uviedli už pri predspracovaní dát, sekvencie API volaní môžu obsahovať užitočnejšie informácie než len jednotlivé výskyty volaní. Vhodná konfigurácia veľkosti n-gramov však závisí od viacerých faktorov a jej výber nie je

[12] jeden riadok textu z datasetu

[13] unikátna zoskupenie znakov z dokumentu, napr. jedno slovo alebo n-gram slov

triviálny. Z tohto dôvodu sa tejto téme budeme podrobnejšie venovať v nasledujúcej časti 5.2.1.

Dôležité je podotknúť, že `CountVectorizer` nie je jedinou možnosťou pre nami požadovanú konverziu. Bežne sa používa aj jeho rozšírenie `TfidfVectorizer`, ktoré okrem frekvencie tokenov zohľadňuje aj ich významnosť v rámci celého korpusu[14]. Dokáže tak lepšie potlačiť vplyv bežne sa vyskytujúcich, ale málo informatívnych volaní, čím môže zlepšiť výsledky modelov. Avšak vzhľadom na spôsob, akým metódy vysvetliteľnosti pracujú so vstupom (konkrétne Anchors a kontrafaktuály), ho nie je možné použiť. `TfidfVectorizer` totiž generuje desatinné hodnoty, preto je výsledok z vysvetliteľných metód vždy nutné spätne previesť do priestoru celých čísiel. Tu však nastáva problém, keďže korektná reverzná transformácia je možná len pri nezmenených vstupoch. V opačnom prípade výsledok prestane zodpovedať skutočným hodnotám a vysvetlenie nebude mať zmysel (napr. výrok „malvér je trojan, ak má API volanie `CreateThread` frekvenciu výskytu presne 4,2069“ je nepoužiteľný). Z tohto dôvodu sme sa, aj za cenu mierne horších výsledkov modelu, rozhodli pre použitie `CountVectorizer`.

Výstupom z `CountVectorizer` sú dáta takmer pripravené na použitie pre trénovanie. Posledným krokom je zakódovanie výstupných tried do číselného formátu, keďže naše klasifikačné algoritmy nedokážu pracovať s textovými výstupmi. Na tento účel sme použili triedu `LabelEncoder`, opäť z knižnice `scikit-learn`. V tomto bode sú naše dáta plne pripravené na trénovanie a môžeme prejsť na ich ladenie.

5.2.1 Ladenie hyperparametrov

Ako sme už spomenuli, zvolenie správnych hyperparametrov pre vektorizáciu je kľúčové na dosiahnutie kvalitných modelov. Keďže máme k dispozícii sekvenčné dáta, máme z nich možnosť vytvoriť n-gramy. N-gramy sú definované ako sekvencia n po sebe idúcich znakov v určitom poradí [48]. Napríklad, zo sekvencie znakov „AB CD EF“, možno vytvoriť tri unigramy[15] „AB“, „CD“, „EF“, dva bigramy[16] „AB CD“, „CD EF“ a jeden trigram[17] „AB CD EF“.

Rovnaký princíp vieme aplikovať aj na sekvenciu API volaní, problémom je však extrémne vysoká dimenzionalita výsledného priestoru. Ak by sme pracovali iba s výskytom volaní (tie predstavujú unigramy), tak už teraz máme k dispozícii 10 513 príznakov. Rozšírenie o bigramy by tento počet navýšilo na 113 392, a ak by sme pridali aj trigramy, narástlo by až na 309 937 rôznych príznakov. Takýto obrovský počet

[14] kolekcia dokumentov
[15] n-gram o veľkosti jedna
[16] n-gram o veľkosti dva
[17] n-gram o veľkosti tri

príznakov by bol extrémne výpočtovo náročný a neefektívny. Preto sa okrem nastavenia rozsahu n-gramov používa aj parameter `max_features`, ktorý ponechá nanajvýš zvolený počet príznakov, zoradených podľa frekvencie výskytu, a ostatné vyradí.

Cieľom testovania v tejto časti bolo nájsť optimálnu kombináciu týchto dvoch hyperparametrov. Testovali sme rozsah n-gramov o veľkosti 1 až 3 a hodnoty `max_features` nastavené na 500, 1 000, 5 000, 10 000 a 15 000. Tieto hyperparametre sme overili na našich troch modeloch, pričom na každom modely sme otestovali všetkých 15 možných kombinácií, čiže dokopy prebehlo 45 rôznych testov. Hyperparametre modelov náhodného lesa a XGBoostu sme ponechali na predvolených hodnotách, zatiaľ čo pre MLP sme zvolili jednoduchú architektúru siete s jednou skrytou vrstvou. Snahou bolo čo najviac izolovať vplyv modelov a posudzovať iba parametre vektorizácie. Podrobnejší popis použitých hyperparametrov modelov sa nachádza v častiach, ktoré sa venujú ladeniu hyperparametrov pre dané modely, v poradí sú to časti 5.3.1, 5.4.1 a 5.5.1. Testovanie prebiehalo pomocou 5-násobnej krížovej validácie. V tabuľkách je každá kombinácia hyperparametrov zapísaná v tvare *n/max*, kde *n* je maximálna veľkosť n-gramu a *max* je maximálny počet príznakov.

Výsledky testovania pre model náhodného lesa (tabuľka č. 3) ukazujú, že unigramy dosahovali veľmi stabilné výsledky naprieč všetkými veľkosťami príznakov, s F1-skóre okolo 0,728. Toto je pomerne prekvapivé, keďže sme predpokladali zlepšenie s rastúcim počtom príznakov nezávisle od zvolenej veľkosti n-gramov. Tento predpoklad sa však naplnil pri použití bigramov a trigramov. Pri nízkom počte príznakov (500) bolo F1-skóre výrazne nižšie (pre bigramy 0,673 a pre trigramy 0,633), no od hodnoty 5 000 príznakov sa skóre stabilizovalo na úrovni okolo 0,73. Z týchto výsledkov nám vyplýva, že RF výrazne nereaguje na širší kontext, ktorý n-gramy poskytujú. A aj to minimálne zlepšenie, ktoré nastalo, sa prejaví až pri podstatne väčšom počte príznakov.

Pre model XGBoost (tabuľka č. 4) sme zaznamenali veľmi podobný trend. Unigramy poskytovali stabilné F1-skóre približne 0,74, pričom nárast počtu príznakov nemal žiaden vplyv na zlepšenie výsledkov. Najlepší výkon bol zaznamenaný pri kombinácii bigramov s minimálne 5 000 príznakmi alebo trigramov s aspoň 10 000 príznakmi, kde F1-skóre dosiahlo hodnotu 0,755. Naopak, pri nízkom počte príznakov bol výkon podstatne horší, kde pre bigramy s 500 príznakmi bolo skóre 0,716 a pre trigramy s 500 príznakmi iba 0,703. Opäť sa zdá, že XGBoost výrazne pozitívne nereaguje na kontext, ktorý n-gramy prinášajú.

Prekvapenie však priniesli výsledky modelu MLP (tabuľka č. 5). Na rozdiel od RF a XGBoost modelov, výsledky pri použití unigramov boli výrazne slabšie. Pre 500 príznakov bolo F1-skóre iba 0,61 a so zvyšujúcim sa počtom príznakov jemne rástol až po skóre 0,627 pre 15 000 príznakov. Významný nárast však nastal pri použití vyšších

Tabuľka 3: Výsledky ladenia hyperparametrov vektorizácie pre model RF

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
1/500	$0{,}825 \pm 0{,}009$	$0{,}769 \pm 0{,}009$	$0{,}705 \pm 0{,}014$	$0{,}728 \pm 0{,}012$
1/1 000	$0{,}825 \pm 0{,}006$	$0{,}767 \pm 0{,}007$	$0{,}702 \pm 0{,}013$	$0{,}725 \pm 0{,}011$
1/5 000	$0{,}829 \pm 0{,}008$	$0{,}765 \pm 0{,}007$	$0{,}708 \pm 0{,}018$	$0{,}729 \pm 0{,}014$
1/10 000	$0{,}829 \pm 0{,}010$	$0{,}765 \pm 0{,}011$	$0{,}708 \pm 0{,}019$	$0{,}729 \pm 0{,}016$
1/15 000	$0{,}828 \pm 0{,}008$	$0{,}764 \pm 0{,}008$	$0{,}706 \pm 0{,}015$	$0{,}728 \pm 0{,}012$
2/500	$0{,}749 \pm 0{,}005$	$0{,}751 \pm 0{,}009$	$0{,}642 \pm 0{,}016$	$0{,}673 \pm 0{,}011$
2/1 000	$0{,}786 \pm 0{,}008$	$0{,}763 \pm 0{,}012$	$0{,}676 \pm 0{,}019$	$0{,}706 \pm 0{,}015$
2/5 000	$0{,}830 \pm 0{,}007$	$0{,}770 \pm 0{,}010$	$0{,}710 \pm 0{,}017$	$0{,}732 \pm 0{,}014$
2/10 000	$0{,}831 \pm 0{,}009$	$0{,}773 \pm 0{,}010$	$0{,}711 \pm 0{,}018$	$0{,}734 \pm 0{,}015$
2/15 000	$0{,}831 \pm 0{,}006$	$0{,}774 \pm 0{,}007$	$0{,}709 \pm 0{,}015$	$0{,}734 \pm 0{,}011$
3/500	$0{,}710 \pm 0{,}006$	$0{,}740 \pm 0{,}012$	$0{,}607 \pm 0{,}012$	$0{,}633 \pm 0{,}010$
3/1 000	$0{,}748 \pm 0{,}006$	$0{,}749 \pm 0{,}010$	$0{,}638 \pm 0{,}018$	$0{,}669 \pm 0{,}013$
3/5 000	$0{,}826 \pm 0{,}007$	$0{,}767 \pm 0{,}008$	$0{,}706 \pm 0{,}013$	$0{,}729 \pm 0{,}011$
3/10 000	$0{,}831 \pm 0{,}009$	$0{,}772 \pm 0{,}011$	$0{,}712 \pm 0{,}020$	$0{,}735 \pm 0{,}016$
3/15 000	$0{,}829 \pm 0{,}007$	$0{,}771 \pm 0{,}010$	$0{,}708 \pm 0{,}016$	$0{,}732 \pm 0{,}013$

n-gramov. Bigramy s 15 000 príznakmi dosiahli F1-skóre 0,721 a trigramy 0,725, čo je v porovnaní s najlepším skóre pre unigramy rozdiel takmer o 0,1. Veľké skokové rasty prebiehali aj v rámci bigramov a trigramov. Najväčší zaznamenaný patril trigramom s 500 príznakmi (0,483) a 1 000 príznakmi (0,628), ktorý predstavoval rozdiel až 0,145. Z výsledkov jasne vyplýva, že modelu MLP výrazne prospieva širší kontext vo vstupných dátach, ktorý n-gramy prinášajú.

Zrejme najväčším prekvapením z celého testovania je stabilita a dosiahnuté skóre pri použití unigramov pre modely náhodného lesa a XGBoostu. Napriek očakávaniam poskytujú aj s minimálnym počtom príznakov takmer rovnaký výkon ako podstatne zložitejšie konfigurácie. Rozdiel v F1-skóre je len 0,007 pre RF a 0,02 pre XGBoost. Z praktického hľadiska by sa tak zdala byť najefektívnejšia kombinácia unigramov a 500 príznakov, kvôli nízkym výpočtovým nárokom a veľkou rýchlosťou spracovania. Výsledky MLP modelu sa však výrazne líšia. Pre unigramy a ľubovoľný počet príznakov bolo dosiahnuté F1-skóre podstatne nižšie. Zlepšenie sme zaznamenali až pri pridaní bigramov a trigramov, a to s aspoň 1 000 príznakmi. Pridaná hodnota n-gramov a väčšieho počtu príznakov je teda očividná.

Tabuľka 4: Výsledky ladenia hyperparametrov vektorizácie pre model XGBoost

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
1/500	$0{,}832 \pm 0{,}007$	$0{,}752 \pm 0{,}013$	$0{,}721 \pm 0{,}015$	$0{,}734 \pm 0{,}014$
1/1 000	$0{,}838 \pm 0{,}006$	$0{,}759 \pm 0{,}009$	$0{,}729 \pm 0{,}015$	$0{,}741 \pm 0{,}012$
1/5 000	$0{,}838 \pm 0{,}006$	$0{,}756 \pm 0{,}009$	$0{,}725 \pm 0{,}016$	$0{,}738 \pm 0{,}012$
1/10 000	$0{,}837 \pm 0{,}006$	$0{,}755 \pm 0{,}012$	$0{,}725 \pm 0{,}016$	$0{,}738 \pm 0{,}014$
1/15 000	$0{,}837 \pm 0{,}006$	$0{,}755 \pm 0{,}012$	$0{,}725 \pm 0{,}017$	$0{,}738 \pm 0{,}014$
2/500	$0{,}792 \pm 0{,}006$	$0{,}734 \pm 0{,}005$	$0{,}708 \pm 0{,}009$	$0{,}716 \pm 0{,}007$
2/1 000	$0{,}798 \pm 0{,}007$	$0{,}740 \pm 0{,}009$	$0{,}718 \pm 0{,}014$	$0{,}724 \pm 0{,}011$
2/5 000	$0{,}847 \pm 0{,}007$	$0{,}774 \pm 0{,}010$	$0{,}743 \pm 0{,}015$	$0{,}756 \pm 0{,}012$
2/10 000	$0{,}846 \pm 0{,}006$	$0{,}773 \pm 0{,}011$	$0{,}741 \pm 0{,}015$	$0{,}755 \pm 0{,}013$
2/15 000	$0{,}845 \pm 0{,}007$	$0{,}772 \pm 0{,}012$	$0{,}741 \pm 0{,}018$	$0{,}755 \pm 0{,}015$
3/500	$0{,}778 \pm 0{,}007$	$0{,}723 \pm 0{,}007$	$0{,}695 \pm 0{,}011$	$0{,}703 \pm 0{,}009$
3/1 000	$0{,}794 \pm 0{,}005$	$0{,}738 \pm 0{,}008$	$0{,}709 \pm 0{,}013$	$0{,}718 \pm 0{,}009$
3/5 000	$0{,}838 \pm 0{,}004$	$0{,}761 \pm 0{,}007$	$0{,}731 \pm 0{,}010$	$0{,}743 \pm 0{,}007$
3/10 000	$0{,}847 \pm 0{,}005$	$0{,}773 \pm 0{,}011$	$0{,}741 \pm 0{,}015$	$0{,}755 \pm 0{,}012$
3/15 000	$0{,}847 \pm 0{,}006$	$0{,}774 \pm 0{,}010$	$0{,}743 \pm 0{,}015$	$0{,}756 \pm 0{,}012$

Na základe výsledkov z našich experimentov sa preto zdá byť najlepším riešením použitie odlišných hyperparametrov vektorizácie pre rôzne modely. Keďže je ale jedným z hlavných cieľov našej práce porovnať výstupy vysvetliteľnosti medzi modelmi, je preto žiadúce, aby mali všetky klasifikátory k dispozícii rovnaké dáta na vstupe. Navyše, neskôr sme narazili na praktické obmedzenie pre počet príznakov, ktoré zvládnu vysvetľujúce metódy spracovať. Tento problém je detailnejšie rozobratý v časti 6.2, dôsledkom je však maximálny počet príznakov 1 000. Výber hyperparametrov si preto vyžadoval dôkladné zváženie všetkých zistení. Nakoniec sme sa rozhodli pre maximálnu veľkosť n-gramu 2 (unigramy a bigramy) a maximálny počet príznakov 1 000. Tieto hyperparametre boli vybrané najmä z dôvodu spomínanej praktickej limitácie na počet príznakov a s ohľadom na výsledky testovania pre model MLP, ktoré pri 1 000 príznakoch dosahujú najlepšie skóre práve pre bigramy. Pre túto konfiguráciu parametrov sme vykonali ešte raz testovanie na všetkých troch modeloch, ale tentokrát už s použitím 10-násobnej krížovej validácie pre spoľahlivejšie výsledky. Výsledky modelov sú spoločne uvedené v tabuľke 6 a predstavujú referenčné hodnoty, ktoré sa v nasledujúcich častiach pokúsime prekonať.

Tabuľka 5: Výsledky ladenia hyperparametrov vektorizácie pre model MLP

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
1/500	$0{,}766 \pm 0{,}006$	$0{,}631 \pm 0{,}009$	$0{,}604 \pm 0{,}017$	$0{,}610 \pm 0{,}013$
1/1 000	$0{,}770 \pm 0{,}003$	$0{,}634 \pm 0{,}004$	$0{,}603 \pm 0{,}009$	$0{,}609 \pm 0{,}011$
1/5 000	$0{,}777 \pm 0{,}012$	$0{,}643 \pm 0{,}016$	$0{,}619 \pm 0{,}025$	$0{,}623 \pm 0{,}024$
1/10 000	$0{,}778 \pm 0{,}011$	$0{,}638 \pm 0{,}015$	$0{,}621 \pm 0{,}021$	$0{,}622 \pm 0{,}021$
1/15 000	$0{,}780 \pm 0{,}006$	$0{,}645 \pm 0{,}013$	$0{,}625 \pm 0{,}014$	$0{,}627 \pm 0{,}011$
2/500	$0{,}724 \pm 0{,}006$	$0{,}629 \pm 0{,}009$	$0{,}587 \pm 0{,}017$	$0{,}594 \pm 0{,}013$
2/1 000	$0{,}754 \pm 0{,}003$	$0{,}667 \pm 0{,}009$	$0{,}639 \pm 0{,}013$	$0{,}646 \pm 0{,}006$
2/5 000	$0{,}826 \pm 0{,}006$	$0{,}729 \pm 0{,}010$	$0{,}706 \pm 0{,}010$	$0{,}715 \pm 0{,}009$
2/10 000	$0{,}828 \pm 0{,}007$	$0{,}732 \pm 0{,}013$	$0{,}711 \pm 0{,}011$	$0{,}720 \pm 0{,}011$
2/15 000	$0{,}828 \pm 0{,}010$	$0{,}733 \pm 0{,}014$	$0{,}712 \pm 0{,}014$	$0{,}721 \pm 0{,}014$
3/500	$0{,}655 \pm 0{,}009$	$0{,}597 \pm 0{,}016$	$0{,}475 \pm 0{,}017$	$0{,}483 \pm 0{,}012$
3/1 000	$0{,}740 \pm 0{,}004$	$0{,}651 \pm 0{,}006$	$0{,}620 \pm 0{,}011$	$0{,}628 \pm 0{,}007$
3/5 000	$0{,}818 \pm 0{,}005$	$0{,}719 \pm 0{,}009$	$0{,}699 \pm 0{,}013$	$0{,}707 \pm 0{,}010$
3/10 000	$0{,}830 \pm 0{,}005$	$0{,}735 \pm 0{,}007$	$0{,}713 \pm 0{,}011$	$0{,}722 \pm 0{,}009$
3/15 000	$0{,}831 \pm 0{,}008$	$0{,}737 \pm 0{,}009$	$0{,}716 \pm 0{,}015$	$0{,}725 \pm 0{,}011$

Tabuľka 6: Výsledky modelov pre finálne hyperparametre vektorizácie

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
RF(2/1 000)	$0{,}790 \pm 0{,}007$	$0{,}769 \pm 0{,}012$	$0{,}683 \pm 0{,}016$	$0{,}712 \pm 0{,}014$
XGB(2/1 000)	$0{,}799 \pm 0{,}008$	$0{,}742 \pm 0{,}014$	$0{,}720 \pm 0{,}017$	$0{,}727 \pm 0{,}015$
MLP(2/1 000)	$0{,}764 \pm 0{,}008$	$0{,}686 \pm 0{,}009$	$0{,}661 \pm 0{,}020$	$0{,}668 \pm 0{,}014$

5.3 Náhodný les

Na vytvorenie modelu RF sme sa z dôvodu nevyváženosti dát rozhodli použiť implementáciu `BalancedRandomForestClassifier` z Python knižnice `imbalanced-learn`. Tá je špeciálne navrhnutá pre veľmi nevyvážené datasety. Od klasickej implementácie náhodného lesa (podkapitola 2.1) sa líši primárne spôsobom vytvárania bootstrap vzoriek pre jednotlivé stromy.

Namiesto náhodného výberu z celého datasetu a zachovania pôvodnej distribúcie tried, poskytuje každému stromu vyváženú podmnožinu dát [49]. Najprv sa určí počet vzoriek v najmenej početnej triede. Následne sa tento počet vzoriek náhodne vyberie z každej triedy. Vznikne tak vyvážená množina, ktorá sa použije na trénovanie jedného

rozhodovacieho stromu. Tento proces sa opakuje zvlášť pre každý strom v lese, vďaka čomu každý strom vidí inú vyváženú vzorku z pôvodných dát. Na rozdiel od klasického *undersampling* neprichádzame trvalo o žiadne dáta, keďže každý strom môže pracovať s inou kombináciou vzoriek z väčšinových tried.

5.3.1 Ladenie hyperparametrov

Ako sme uviedli v podkapitole 2.1, algoritmus náhodného lesa dosahuje často veľmi dobré výsledky aj s predvolenými hodnotami hyperparametrov, ktoré autori uvádzajú v oficiálnej dokumentácii [49]. Tieto predvolené nastavenia sme použili aj pri ladení hyperparametrov vektorizácie pre model RF.

Konkrétne hodnoty si teraz uvedieme. Počet rozhodovacích stromov (`n_estimators`) bol nastavený na 100. Maximálna hĺbka stromov (`max_depth`) nebola obmedzená, čo umožňuje stromom rásť, až kým listový uzol neobsahuje minimálny definovaný počet vzoriek. Minimálny počet vzoriek potrebných na rozdelenie uzla (`min_samples_split`) mal hodnotu 2, zatiaľ čo minimálny počet vzoriek v listovom uzle (`min_samples_leaf`) bol nastavený na 1. Počet príznakov použitých pri rozdelení uzla bol riadený pravidlom `max_features = sqrt`, čo spôsobuje, že pri každom rozdelení sa náhodne vyberie odmocnina z celkového počtu príznakov. S týmito parametrami sme pri 10-násobnej krížovej validácii dosiahli základné F1-skóre $0{,}712 \pm 0{,}014$, ktoré použijeme ako referenčnú hodnotu pre porovnanie na konci ladenia. V tejto časti sa teraz ladením týchto hyperparametrov pokúsime toto skóre čo najviac zvýšiť a tým zlepšiť výkon modelu.

Podľa analýzy v štúdii [50], počet stromov nie je potrebné detailne ladiť. Odporúča sa ho nastaviť na čo najvyššiu hodnotu, ktorú výpočtové schopnosti dovoľujú. Keďže sú rozhodovacie stromy trénované oddelene, neexistuje riziko pretrénovania ani s veľkým počtom stromov. V praxi však od určitého bodu prináša väčší počet stromov len zanedbateľné zlepšenia. Z tohto dôvodu sme sa rozhodli tento parameter doladiť ako posledný, po určení ostatných hyperparametrov. Ponecháme ho preto zatiaľ na predvolenej hodnote kvôli urýchleniu trénovania.

Ako vyplýva z práce [12], najväčší potenciál na zlepšenie výsledkov modelu má parameter `max_features`. Ostatné parametre ako `min_samples_split` a `min_samples_leaf` majú menší význam, keďže ich predvolené hodnoty majú dostatočnú schopnosť zachytiť zložité vzory v dátach. Ich úprava sa odporúča najmä pri potrebe znížiť nároky na výpočtový výkon, nie kvôli zlepšeniu výsledkov modelu. Hyperparameter `max_depth` bol síce spomenutý len okrajovo, ale intuitívne sa dá predpokladať, že vyššie hodnoty umožnia zachytiť aj komplexnejšie vzorce, no za cenu vyšších výpočtových nárokov.

Na základe týchto zistení sme sa rozhodli ladiť hyperparametre `max_depth` a `max_features`. Na tento účel bola použitá knižnica `optuna`, ktorá využíva Bayesovskú

optimalizáciu na efektívne prehľadávanie priestoru hyperparametrov. Rozsah hľadania sme pre `max_depth` špecifikovali od 11 po 31 s krokom 2. Horná hranica bola zvolená kvôli obmedzeniam GPU verzie implementácie knižnice SHAP, ktorá podporuje stromy s hĺbkou nanajvýš práve 31. Pre parameter `max_features` sme testovali celý rozsah od 0,01 po 1,0 s krokom 0,01, pričom tieto hodnoty reprezentujú percentuálny podiel z celkového počtu príznakov. Spolu sme vykonali 20 testov a použitá bola 5-násobná krížová validácia. Hyperparametre sú v tabuľke zapisované vo formáte `max_depth/max_features`. Pre lepšiu čitateľnosť boli jednotlivé merania zoradené vzostupne podľa hodnôt hyperparametrov.

Tabuľka 7: Výsledky ladenia hyperparametrov `max_depth` a `max_features` pre model RF

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`11/0,32`	$0{,}726 \pm 0{,}007$	$0{,}729 \pm 0{,}005$	$0{,}616 \pm 0{,}009$	$0{,}632 \pm 0{,}006$
`13/0,06`	$0{,}699 \pm 0{,}007$	$0{,}731 \pm 0{,}008$	$0{,}596 \pm 0{,}014$	$0{,}610 \pm 0{,}010$
`17/0,46`	$0{,}778 \pm 0{,}006$	$0{,}761 \pm 0{,}005$	$0{,}666 \pm 0{,}013$	$0{,}695 \pm 0{,}008$
`17/0,79`	$0{,}774 \pm 0{,}007$	$0{,}754 \pm 0{,}008$	$0{,}662 \pm 0{,}015$	$0{,}689 \pm 0{,}011$
`17/0,81`	$0{,}774 \pm 0{,}007$	$0{,}754 \pm 0{,}007$	$0{,}661 \pm 0{,}015$	$0{,}689 \pm 0{,}011$
`21/0,19`	$0{,}785 \pm 0{,}006$	$0{,}766 \pm 0{,}007$	$0{,}674 \pm 0{,}012$	$0{,}704 \pm 0{,}009$
`21/0,69`	$0{,}784 \pm 0{,}008$	$0{,}764 \pm 0{,}013$	$0{,}671 \pm 0{,}020$	$0{,}701 \pm 0{,}017$
`23/0,26`	$0{,}787 \pm 0{,}008$	$0{,}767 \pm 0{,}011$	$0{,}678 \pm 0{,}016$	$0{,}707 \pm 0{,}013$
`23/0,44`	$0{,}787 \pm 0{,}009$	$0{,}767 \pm 0{,}013$	$0{,}677 \pm 0{,}019$	$0{,}707 \pm 0{,}016$
`25/0,70`	$0{,}786 \pm 0{,}008$	$0{,}762 \pm 0{,}010$	$0{,}676 \pm 0{,}018$	$0{,}705 \pm 0{,}015$
`27/0,33`	$0{,}788 \pm 0{,}007$	$0{,}766 \pm 0{,}010$	$0{,}679 \pm 0{,}016$	$0{,}708 \pm 0{,}013$
`27/0,44`	$0{,}787 \pm 0{,}007$	$0{,}765 \pm 0{,}009$	$0{,}676 \pm 0{,}015$	$0{,}707 \pm 0{,}013$
`27/0,79`	$0{,}786 \pm 0{,}007$	$0{,}760 \pm 0{,}011$	$0{,}677 \pm 0{,}015$	$0{,}705 \pm 0{,}013$
`29/0,63`	$0{,}787 \pm 0{,}008$	$0{,}765 \pm 0{,}011$	$0{,}678 \pm 0{,}015$	$0{,}707 \pm 0{,}014$
`29/0,98`	$0{,}786 \pm 0{,}008$	$0{,}759 \pm 0{,}012$	$0{,}675 \pm 0{,}018$	$0{,}703 \pm 0{,}016$
`31/0,53`	$0{,}788 \pm 0{,}008$	$0{,}765 \pm 0{,}012$	$0{,}680 \pm 0{,}014$	$0{,}709 \pm 0{,}013$
`31/0,54`	$0{,}788 \pm 0{,}007$	$0{,}765 \pm 0{,}012$	$0{,}680 \pm 0{,}015$	$0{,}709 \pm 0{,}013$
`31/0,56`	$0{,}787 \pm 0{,}007$	$0{,}763 \pm 0{,}012$	$0{,}677 \pm 0{,}015$	$0{,}707 \pm 0{,}013$
`31/0,57`	$0{,}788 \pm 0{,}007$	$0{,}764 \pm 0{,}011$	$0{,}679 \pm 0{,}015$	$0{,}708 \pm 0{,}012$
`31/0,95`	$0{,}786 \pm 0{,}008$	$0{,}759 \pm 0{,}011$	$0{,}676 \pm 0{,}016$	$0{,}704 \pm 0{,}014$

Výsledky jednotlivých kombinácií sú zobrazené v tabuľke 7. Podľa očakávaní, so zvyšujúcou sa hĺbkou stromu rástlo aj F1-skóre. Výrazný vplyv však malo len po hĺbku 21, po tejto hodnote sa už výkon zlepšoval len minimálne a dá sa predpokladať, že

ho ovplyvňovala skôr hodnota `max_features`. Tento vplyv ide najlepšie pozorovať na posledných piatich riadkoch v tabuľke, kedy sa hĺbka stromu už nemenila.

Z tabuľky tiež vidno, že viacero kombinácií dosiahlo veľmi podobné výsledky. Spomedzi nich sme vybrali troch kandidátov a to kombinácie `23/0,26` (0,707), `27/0,33` (0,708) a `31/0,53` (0,709). Napriek tomu, že posledná kombinácia má o trochu vyššie skóre, jej maximálna hĺbka je už na hranici. Z praktického hľadiska preferujeme modely s čo najnižšou hĺbkou, keďže tá výrazne vplýva na výpočtový čas modelu. Tieto tri kombinácie boli následne opäť otestované, tentokrát však s 10-násobnou krížovou validáciou. Pred výberom sme chceli totiž overiť ich presnosť s väčšou spoľahlivosťou. Výsledky tohto testovania sú zhrnuté v tabuľke 8.

Tabuľka 8: Porovnanie troch najúspešnejších kombinácií hyperparametrov `max_depth` a `max_features` pre model RF

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`23/0,26`	$0{,}791 \pm 0{,}008$	$0{,}773 \pm 0{,}012$	$0{,}684 \pm 0{,}016$	$0{,}714 \pm 0{,}013$
`27/0,33`	$0{,}792 \pm 0{,}007$	$0{,}773 \pm 0{,}011$	$0{,}686 \pm 0{,}014$	$0{,}716 \pm 0{,}012$
`31/0,53`	$0{,}792 \pm 0{,}008$	$0{,}770 \pm 0{,}013$	$0{,}686 \pm 0{,}013$	$0{,}715 \pm 0{,}013$

Použitie 10-násobnej krížovej validácie potvrdilo výsledky z predchádzajúcich testov. Pre všetky tri kombinácie sme dokonca zaznamenali mierne lepšie výsledky. Priemerné zlepšenie F1-skóre bolo približne 0,007. Najlepší výsledok, aj keď iba s minimálnym rozdielom, dosiahla konfigurácia `27/0,33` s F1-skóre $0{,}716 \pm 0{,}012$. Navyše, hĺbka stromu je pod definovaným maximom, čo prispieva k efektívnejšiemu trénovaniu modelu. Vybrali sme si preto túto kombináciu hyperparametrov.

Na záver zostávala už len optimalizácia hyperparametra `n_estimators`, čiže počtu rozhodovacích stromov. Testovanie sme vykonali v rozsahu od 150 do 300 stromov s krokom 50, pričom pre porovnanie bol zahrnutý aj výsledok pre 100 stromov. Vzhľadom na to, že išlo o poslednú fázu ladenia modelu, žiadúce bolo zabezpečiť čo najvyššiu spoľahlivosť výsledkov. Opäť sme preto použili 10-násobnú krížovú validáciu.

Výsledky testovania sú uvedené v tabuľke 9. Ako možno vidieť, F1-skóre sa s rastúcim počtom stromov mierne zlepšovalo. Z pôvodnej hodnoty $0{,}715 \pm 0{,}013$ pri 100 stromoch na $0{,}718 \pm 0{,}013$ pri 300 stromoch. Ide však len o marginálne zlepšenie o 0,003 bodu, zatiaľ čo čas potrebný na trénovanie modelu sa zvýšil viac než trojnásobne. Z praktického hľadiska sme preto ako kompromis zvolili 200 stromov, pre ktoré sme zaznamenali zlepšenie skóre o 0,002, no s výrazne nižšou výpočtovou náročnosťou.

Tabuľka 9: Výsledky ladenia hyperparametra `n_estimators` pre model RF

Počet stromov	Správnosť	Presnosť	Senzitivita	F1-skóre
100	$0{,}792 \pm 0{,}008$	$0{,}770 \pm 0{,}013$	$0{,}686 \pm 0{,}013$	$0{,}715 \pm 0{,}013$
150	$0{,}792 \pm 0{,}007$	$0{,}773 \pm 0{,}014$	$0{,}686 \pm 0{,}014$	$0{,}715 \pm 0{,}013$
200	$0{,}792 \pm 0{,}007$	$0{,}775 \pm 0{,}013$	$0{,}687 \pm 0{,}013$	$0{,}717 \pm 0{,}012$
250	$0{,}793 \pm 0{,}008$	$0{,}775 \pm 0{,}013$	$0{,}688 \pm 0{,}014$	$0{,}717 \pm 0{,}013$
300	$0{,}793 \pm 0{,}007$	$0{,}775 \pm 0{,}013$	$0{,}689 \pm 0{,}014$	$0{,}718 \pm 0{,}013$

Týmto považujeme proces ladenia modelu náhodného lesa za ukončený. Finálny model dosiahol F1-skóre $0{,}717 \pm 0{,}012$, čo predstavuje zlepšenie o 0,005 bodu oproti referenčnému skóre $0{,}712 \pm 0{,}014$. Nejedná sa o veľký nárast, je však v súlade s analýzou zo štúdie [12]. Výsledné optimalizované hyperparametre teda sú 200 rozhodovacích stromov, maximálna hĺbka stromu 27 a podiel použitých príznakov pri rozdelení uzla ako 0,33. Ostatné nastavenia sme ponechali na predvolených hodnotách.

5.4 XGBoost

Na implementáciu klasifikátora XGBoost sme použili triedu `XGBClassifier` z Python knižnice `xgboost`. Táto trieda funguje ako `wrapper` nad základnou implementáciou algoritmu XGBoost a je prispôsobená na jednoduchú integráciu s nástrojmi knižnice `scikit-learn`. Umožňuje tiež jednoduchšie a intuitívnejšie ovládanie, pričom zachováva dostatočnú úroveň kontroly nad nastaviteľnými parametrami modelu.

`XGBClassifier` taktiež podporuje techniku skorého zastavenia, ktorá zabezpečuje predčasné ukončenie trénovania v prípade, že sa sledovaná metrika na validačnej množine po určitý počet iterácií nezlepšuje. Poskytované možnosti sú však pomerne obmedzené a neumožňujú priamo použiť F1-skóre ako metriku. Bolo preto nutné implementovať vlastnú metódu na výpočet F1-skóre, ktorá bola následne integrovaná do trénovacieho procesu klasifikátora.

5.4.1 Ladenie hyperparametrov

Algoritmus XGBoost dokáže síce docieliť vysokú predikčnú presnosť, no tá vo veľkej miere závisí od správne nastavených hyperparametrov, ktorých je navyše veľké množstvo. V tejto časti si popíšeme proces, akým sme postupovali pri hľadaní optimálnych parametrov. Za referenčné F1-skóre, ktoré sa pokúsime prekonať, sme zobrali výsledok 10-násobnej krížovej validácie modelu s predvolenými hodnotami, tak ako sú uvedené v oficiálnej

dokumentácii [51]. Dosiahli sme výsledok $0{,}727 \pm 0{,}015$. Tento model sme zároveň použili na ladenie hyperparametrov vektorizácie v časti 5.2.1.

Predvolené hyperparametre si teraz popíšeme. Parameter maximálnej hĺbky stromu (`max_depth`) bol nastavený na hodnotu 6. Hlbšie stromy môžu modelu umožniť zachytiť komplexnejšie vzory, no zároveň zvyšujú riziko preučenia a nároky na pamäť. Ďalší parameter je `min_child_weight` a jeho predvolenou hodnotou je 1. Udáva minimálnu sumu váh v uzle, ktorú musí spĺňať, aby mohol byť rozdelený. Vyššie hodnoty znižujú pravdepodobnosť preučenia, no môžu zhoršiť výsledky modelu.

Ďalšie dva parametre, `subsample` a `colsample_bytree`, riadia mieru náhodnosti počas trénovania. Parameter `subsample` určuje, aký podiel trénovacích dát sa použije pri trénovaní každého stromu. Predvolená hodnota je 1 a znamená použitie všetkých dát. Parameter `colsample_bytree` zasa definuje podiel príznakov, ktorý sa použije pri vytváraní jednotlivých stromov, pričom predvolená hodnota je opäť 1 a spôsobí použitie všetkých príznakov. Zníženie týchto hodnôt môže pomôcť znížiť riziko preučenia a zvýšiť robustnosť modelu voči korelovaným alebo nerelevantným príznakom.

Parameter rýchlosť učenia (`learning_rate`) má predvolenú hodnotu 0,3 a riadi veľkosť aktualizácií váh modelu po každom kole boostingu. Nižšia hodnota zvyčajne vedie k lepšej generalizácii, no vyžaduje väčší počet stromov. Tým sa dostávame k poslednému parametru, počet rozhodovacích stromov (`n_estimators`), ktorého predvolená hodnota je 100. Kombinácia veľkého počtu stromov s nižšou hodnotou rýchlosti učenia často vedie k lepším výsledkom, no za cenu značných výpočtových nárokov.

Ladenie všetkých hyperparametrov naraz by bolo výpočtovo náročné a časovo neefektívne, keďže by si vyžadovalo obrovské množstvo testov. Namiesto toho sme použili stratégiu postupného ladenia po skupinách. Hyperparametre sme rozdelili do troch skupín [52]:

1. **max_depth a min_child_weight**: ovplyvňujú komplexnosť modelu
2. **subsample a colsample_bytree**: riadia náhodnosť a robustnosť modelu
3. **n_estimators a learning_rate**: ovplyvňujú rýchlosť a stabilitu učenia

Na začiatku sme model nastavili na predvolené hodnoty. Následne optimalizujeme prvú skupinu parametrov. Po nájdení najlepšej kombinácie tieto hodnoty aplikujeme na model a pokračujeme ladením ďalšej skupiny. Tento proces opakujeme, až kým nenájdeme parametre tretej skupiny. V tomto bode máme všetky hyperparametre a ladenie sa môže skončiť.

Na optimalizáciu sme opäť použili knižnicu `optuna` v kombinácii s 5-násobnou krížovou validáciou. V prvej fáze sme ladili parametre `max_depth` v rozsahu od 6 do 30

s krokom 2 a `min_child_weight` v rozsahu od 1 do 9 s krokom 2. Pre túto kombináciu sme vykonali 15 testov. Výsledky sú uvedené v tabuľke 10. Kombinácie parametrov sú zapísané v tvare `max_depth/min_child_weight` a zoradené vzostupne.

Tabuľka 10: Výsledky ladenia hyperparametrov `max_depth` a `min_child_weight` pre model XGBoost

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`6/1`	$0{,}798 \pm 0{,}007$	$0{,}740 \pm 0{,}009$	$0{,}718 \pm 0{,}014$	$0{,}724 \pm 0{,}011$
`6/3`	$0{,}795 \pm 0{,}007$	$0{,}733 \pm 0{,}012$	$0{,}711 \pm 0{,}015$	$0{,}718 \pm 0{,}013$
`8/1`	$0{,}800 \pm 0{,}006$	$0{,}743 \pm 0{,}008$	$0{,}720 \pm 0{,}011$	$0{,}727 \pm 0{,}010$
`10/1`	$0{,}799 \pm 0{,}006$	$0{,}740 \pm 0{,}009$	$0{,}720 \pm 0{,}013$	$0{,}726 \pm 0{,}011$
`12/5`	$0{,}799 \pm 0{,}006$	$0{,}739 \pm 0{,}009$	$0{,}720 \pm 0{,}012$	$0{,}726 \pm 0{,}011$
`12/9`	$0{,}797 \pm 0{,}008$	$0{,}739 \pm 0{,}009$	$0{,}717 \pm 0{,}014$	$0{,}724 \pm 0{,}012$
`14/5`	$0{,}799 \pm 0{,}006$	$0{,}742 \pm 0{,}015$	$0{,}716 \pm 0{,}009$	$0{,}725 \pm 0{,}011$
`14/7`	$0{,}798 \pm 0{,}006$	$0{,}738 \pm 0{,}007$	$0{,}719 \pm 0{,}013$	$0{,}725 \pm 0{,}011$
`18/1`	$0{,}799 \pm 0{,}008$	$0{,}747 \pm 0{,}017$	$0{,}712 \pm 0{,}013$	$0{,}725 \pm 0{,}013$
`20/3`	$0{,}798 \pm 0{,}007$	$0{,}742 \pm 0{,}012$	$0{,}711 \pm 0{,}014$	$0{,}722 \pm 0{,}010$
`22/7`	$0{,}798 \pm 0{,}005$	$0{,}742 \pm 0{,}012$	$0{,}712 \pm 0{,}015$	$0{,}723 \pm 0{,}010$
`24/3`	$0{,}800 \pm 0{,}008$	$0{,}749 \pm 0{,}014$	$0{,}707 \pm 0{,}010$	$0{,}724 \pm 0{,}011$
`24/7`	$0{,}797 \pm 0{,}006$	$0{,}738 \pm 0{,}015$	$0{,}712 \pm 0{,}005$	$0{,}721 \pm 0{,}008$
`30/5`	$0{,}798 \pm 0{,}007$	$0{,}745 \pm 0{,}014$	$0{,}711 \pm 0{,}017$	$0{,}724 \pm 0{,}012$
`30/9`	$0{,}796 \pm 0{,}006$	$0{,}741 \pm 0{,}012$	$0{,}705 \pm 0{,}014$	$0{,}719 \pm 0{,}010$

Z výsledkov vyplýva, že najlepšia konfigurácia je `8/1`, ktorá dosiahla F1-skóre 0,727. Podobné výsledky mali aj konfigurácie `10/1` a `12/5`. To naznačuje, že optimálna hĺbka stromu je medzi 8 a 12. Pri vyšších hodnotách `max_depth` už dochádzalo skôr k stagnácii alebo miernemu poklesu skóre. Zároveň možno pozorovať pozitívny vplyv nižších hodnôt parametra `min_child_weight`. Dve z troch najlepších konfigurácií majú hodnotu 1. Tento trend sa potvrdzuje aj pri porovnaní dvojíc konfigurácií, kde je hĺbka stromu rovnaká a mení sa len `min_child_weight`. Vo všetkých takýchto prípadoch dosahuje nižšia hodnota `min_child_weight` lepšie alebo prinajmenšom porovnateľné výsledky než vyššia. Napokon, sme pre túto skupinu vybrali konfiguráciu `8/1` a môžeme pokračovať na ďalšiu dvojicu hyperparametrov.

V druhej fáze budeme optimalizovať parametre `subsample` a `colsample_bytree`. Interval hodnôt sme si pre oba parametre definovali od 0,3 do 1,0 s krokom 0,05. Dokopy sme vykonali 15 testov s 5-násobnou krížovou validáciou. Výsledky testovania je možné

vidieť v tabuľke 11. Konkrétna konfigurácia hyperparametrov je v nej vyznačená vo formáte `subsample/colsample_bytree`.

Tabuľka 11: Výsledky ladenia hyperparametrov `subsample` a `colsample_bytree` pre model XGBoost

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`0,05/0,90`	$0{,}796 \pm 0{,}006$	$0{,}737 \pm 0{,}009$	$0{,}714 \pm 0{,}014$	$0{,}722 \pm 0{,}012$
`0,35/0,30`	$0{,}790 \pm 0{,}005$	$0{,}726 \pm 0{,}005$	$0{,}707 \pm 0{,}011$	$0{,}713 \pm 0{,}007$
`0,35/0,50`	$0{,}794 \pm 0{,}006$	$0{,}732 \pm 0{,}011$	$0{,}710 \pm 0{,}014$	$0{,}717 \pm 0{,}012$
`0,50/0,60`	$0{,}796 \pm 0{,}006$	$0{,}737 \pm 0{,}010$	$0{,}717 \pm 0{,}012$	$0{,}723 \pm 0{,}011$
`0,55/0,85`	$0{,}794 \pm 0{,}006$	$0{,}734 \pm 0{,}008$	$0{,}712 \pm 0{,}012$	$0{,}719 \pm 0{,}010$
`0,65/0,40`	$0{,}796 \pm 0{,}007$	$0{,}736 \pm 0{,}010$	$0{,}715 \pm 0{,}010$	$0{,}722 \pm 0{,}010$
`0,70/0,45`	$0{,}798 \pm 0{,}006$	$0{,}742 \pm 0{,}007$	$0{,}720 \pm 0{,}007$	$0{,}727 \pm 0{,}006$
`0,70/1,00`	$0{,}797 \pm 0{,}008$	$0{,}736 \pm 0{,}011$	$0{,}714 \pm 0{,}012$	$0{,}721 \pm 0{,}011$
`0,80/0,80`	$0{,}797 \pm 0{,}007$	$0{,}737 \pm 0{,}012$	$0{,}718 \pm 0{,}014$	$0{,}724 \pm 0{,}013$
`0,80/0,85`	$0{,}795 \pm 0{,}008$	$0{,}733 \pm 0{,}011$	$0{,}711 \pm 0{,}015$	$0{,}718 \pm 0{,}013$
`0,85/0,50`	$0{,}797 \pm 0{,}006$	$0{,}739 \pm 0{,}006$	$0{,}717 \pm 0{,}011$	$0{,}724 \pm 0{,}009$
`0,95/0,30`	$0{,}797 \pm 0{,}007$	$0{,}738 \pm 0{,}007$	$0{,}712 \pm 0{,}012$	$0{,}720 \pm 0{,}009$
`0,95/0,50`	$0{,}800 \pm 0{,}008$	$0{,}741 \pm 0{,}011$	$0{,}721 \pm 0{,}014$	$0{,}727 \pm 0{,}012$
`1,00/0,70`	$0{,}798 \pm 0{,}007$	$0{,}737 \pm 0{,}010$	$0{,}713 \pm 0{,}013$	$0{,}721 \pm 0{,}011$
`1,00/1,00`	$0{,}800 \pm 0{,}006$	$0{,}743 \pm 0{,}008$	$0{,}720 \pm 0{,}011$	$0{,}727 \pm 0{,}010$

Pri ladení hyperparametrov `subsample` a `colsample_bytree` sa ako najúspešnejšie ukázali konfigurácie `0,70/0,45`, `0,95/0,50` a `1,00/1,00`. Všetky tri dosiahli najvyššie F1-skóre na úrovni 0,727. Napokon sme si zvolili konfiguráciu `0,70/0,45` a to z dvoch dôvodov. Jej štandardná odchýlka pre F1-skóre je len 0,006, čo ju robí najnižšou v rámci celého testovania a naznačuje vysokú stabilitu. Ďalšou výhodou je aj zníženie výpočtovej záťaže, keďže potrebuje menší počet vzoriek a príznakov na trénovanie jednotlivých stromov.

Následne môžeme prejsť do poslednej fázy, v ktorej budeme ladiť hyperparametre `n_estimators` a `learning_rate`. Pre parameter `n_estimators` sme určili interval od 100 do 500 s krokom 25. Horná hranica bola vybraná kvôli vysokým nárokom na výkon pre väčšie počty stromov. Rozsah hodnôt pre `learning_rate` sme definovali od 0,001 do 0,3 s krokom 0,001. Vykonaných bolo 20 testov s 5-násobnou krížovou validáciou. Výsledky testov sú uvedené v tabuľke 12 a konfigurácia je v tvare `n_estimators/learning_rate`. Opäť boli zoradené vzostupne podľa hodnôt parametrov.

Tabuľka 12: Výsledky ladenia hyperparametrov `n_estimators` a `learning_rate` pre model XGBoost

Konfigurácia	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`100/0,068`	$0{,}796 \pm 0{,}007$	$0{,}740 \pm 0{,}009$	$0{,}710 \pm 0{,}012$	$0{,}719 \pm 0{,}010$
`125/0,201`	$0{,}800 \pm 0{,}008$	$0{,}743 \pm 0{,}011$	$0{,}720 \pm 0{,}015$	$0{,}727 \pm 0{,}013$
`175/0,083`	$0{,}800 \pm 0{,}005$	$0{,}745 \pm 0{,}009$	$0{,}719 \pm 0{,}009$	$0{,}727 \pm 0{,}009$
`175/0,255`	$0{,}797 \pm 0{,}007$	$0{,}739 \pm 0{,}010$	$0{,}716 \pm 0{,}011$	$0{,}724 \pm 0{,}011$
`200/0,230`	$0{,}799 \pm 0{,}009$	$0{,}743 \pm 0{,}011$	$0{,}720 \pm 0{,}015$	$0{,}728 \pm 0{,}014$
`225/0,184`	$0{,}801 \pm 0{,}006$	$0{,}744 \pm 0{,}008$	$0{,}723 \pm 0{,}011$	$0{,}730 \pm 0{,}010$
`225/0,198`	$0{,}800 \pm 0{,}008$	$0{,}745 \pm 0{,}009$	$0{,}723 \pm 0{,}013$	$0{,}730 \pm 0{,}011$
`250/0,141`	$0{,}800 \pm 0{,}007$	$0{,}744 \pm 0{,}010$	$0{,}722 \pm 0{,}011$	$0{,}729 \pm 0{,}009$
`250/0,181`	$0{,}802 \pm 0{,}007$	$0{,}747 \pm 0{,}010$	$0{,}724 \pm 0{,}010$	$0{,}732 \pm 0{,}010$
`275/0,116`	$0{,}801 \pm 0{,}007$	$0{,}745 \pm 0{,}010$	$0{,}723 \pm 0{,}014$	$0{,}730 \pm 0{,}012$
`275/0,167`	$0{,}800 \pm 0{,}007$	$0{,}744 \pm 0{,}008$	$0{,}721 \pm 0{,}013$	$0{,}728 \pm 0{,}011$
`300/0,167`	$0{,}800 \pm 0{,}007$	$0{,}744 \pm 0{,}008$	$0{,}721 \pm 0{,}013$	$0{,}728 \pm 0{,}011$
`325/0,127`	$0{,}800 \pm 0{,}007$	$0{,}744 \pm 0{,}011$	$0{,}721 \pm 0{,}012$	$0{,}728 \pm 0{,}011$
`325/0,144`	$0{,}800 \pm 0{,}008$	$0{,}744 \pm 0{,}011$	$0{,}721 \pm 0{,}015$	$0{,}728 \pm 0{,}012$
`350/0,110`	$0{,}802 \pm 0{,}007$	$0{,}749 \pm 0{,}010$	$0{,}726 \pm 0{,}013$	$0{,}733 \pm 0{,}011$
`350/0,123`	$0{,}801 \pm 0{,}007$	$0{,}746 \pm 0{,}011$	$0{,}724 \pm 0{,}014$	$0{,}731 \pm 0{,}013$
`375/0,164`	$0{,}800 \pm 0{,}009$	$0{,}745 \pm 0{,}012$	$0{,}723 \pm 0{,}017$	$0{,}730 \pm 0{,}014$
`425/0,077`	$0{,}799 \pm 0{,}008$	$0{,}743 \pm 0{,}011$	$0{,}719 \pm 0{,}014$	$0{,}727 \pm 0{,}012$
`450/0,055`	$0{,}801 \pm 0{,}007$	$0{,}747 \pm 0{,}009$	$0{,}722 \pm 0{,}012$	$0{,}730 \pm 0{,}010$
`450/0,139`	$0{,}799 \pm 0{,}008$	$0{,}742 \pm 0{,}015$	$0{,}720 \pm 0{,}016$	$0{,}727 \pm 0{,}015$

Najvyššie F1-skóre dosiahla konfigurácia s parametrami `350/0,110`, konkrétne 0,733. Výsledky takisto potvrdili predpoklad, že kombinácia väčšieho počtu stromov a nižšej hodnoty rýchlosti učenia má zvyčajne pozitívny vplyv na kvalitu predikcie, je však pomerne náročné nájsť správny pomer. Tento trend je možné pozorovať najmä pri konfiguráciách so zhodným počtom stromov ale odlišnou rýchlosťou učenia. Napr. `450/0,139` (0,727) oproti `450/0,055` (0,730), alebo `275/0,116` (0,730) a `275/0,167` (0,728).

Trend je takisto možné pozorovať pre dvojicu `225/0,184` a `250/0,181`, kde zvýšenie počtu stromov pri takmer nezmenenej hodnote `learning_rate` viedlo k miernemu zlepšeniu F1-skóre z 0,730 na 0,732. Na druhej strane, konfigurácie `175/0,083` (0,727) a `425/0,077` (0,727) ukazujú, že tento trend nie je pravidlom, keďže pri takmer totožnej rýchlosti učenia by sme s nárastom počtu stromov očakávali zlepšenie. To sa však

nepreukázalo. Naznačuje to buď zaseknutie v lokálnom minime pri takejto rýchlosti učenia alebo vplyv iných faktorov, ktoré sa nám však z testov nepodarilo identifikovať.

Dokončením poslednej fázy, môžeme ladenie považovať za ukončené. V prvej skupine parametrov dosiahli najlepšie výsledky `max_depth` na hodnote 8 a `min_child_weight` na 1. V rámci druhej skupiny hodnoty 0,7 pre `subsample` a 0,45 pre `colsample_bytree`. A tretia skupina parametrov nastavenie `learning_rate` na úrovni 0,11 a `n_estimators` na hodnote 350.

Pre čo najspoľahlivejší výsledok sme vykonali 10-násobnú krížovú validáciu, ktorá pre túto konfiguráciu priniesla F1-skóre vo výške $0{,}733 \pm 0{,}012$. V porovnaní s referenčným skóre pre predvolené nastavenia parametrov, ktoré dosiahlo $0{,}727 \pm 0{,}015$, sme zaznamenali zlepšenie o 0,006 bodu a tiež zníženie štandardnej odchýlky o 0,003 bodu. Ide len o mierne zlepšenie, čo naznačuje, že predvolené hodnoty poskytovali dobre nastavený základ. Doladením hyperparametrov sme však docielili stabilnejší a o niečo spoľahlivejší model.

5.5 Viacvrstvový perceptrón

Pre implementáciu modelu viacvrstvového perceptrónu sme sa rozhodli použiť Python knižnicu `PyTorch`. Tá poskytuje nízkoúrovňovú kontrolu nad architektúrou neurónovej siete, jej trénovaním aj optimalizáciou, vďaka čomu umožňuje model prispôsobiť podľa našich požiadaviek.

Knižnica `scikit-learn` takisto poskytuje vlastnú implementáciu viacvrstvového perceptrónu v podobe triedy `MLPClassifier`. Tá je jednoduchšia na použitie a má aj prístupnejšie rozhranie, avšak na rozdiel od `PyTorch` nepodporuje využitie GPU. To predstavuje významné obmedzenie, keďže trénovanie neurónových sietí je často veľmi výpočtovo náročné. Tento faktor výrazne ovplyvnil naše rozhodnutie pri výbere knižnice.

5.5.1 Ladenie hyperparametrov

Na začiatok sme vytvorili jednoduchú neurónovú sieť, ktorá sa skladala z jednej skrytej plne prepojenej vrstvy s 256 neurónmi. Za touto vrstvou nasledovala aktivačná funkcia ReLU a regularizačná dropout vrstva s pravdepodobnostnou mierou 0,3. Na trénovanie modelu sme použili dávky *batch* o veľkosti 128 a rýchlosť učenia sme nastavili na 0,001. Implementovali sme tiež techniku skorého zastavenia s limitom 20 iterácií. Aby sme predišli predčasnému ukončeniu trénovania, počet epoch sme nastavili na 300. Pri tejto konfigurácii sme pomocou 10-násobnej krížovej validácie dosiahli F1-skóre $0{,}668 \pm 0{,}014$. Toto skóre nám bude slúžiť ako referenčný výsledok a v tejto časti sa ho ladením

pokúsime prekonať. Tento jednoduchý model sme zároveň použili aj pri optimalizácii hyperparametrov vektorizácie, popísanej v časti 5.2.1.

Podobne ako pri ladení hyperparametrov pre model XGBoost, tak aj tu by bolo ladenie všetkých parametrov naraz veľmi neefektívne a zdĺhavé. Parametre sme sa preto rozhodli rozdeliť do skupín, ktoré budeme postupne optimalizovať. V jednotlivých skupinách sme sa snažili zahrnúť hyperparametre, ktoré spolu priamo súvisia alebo majú najväčší vplyv na podobné aspekty trénovania modelu. Vytvorili sme tri skupiny:

1. **veľkosť dávky a rýchlosť učenia**: ovplyvňujú rýchlosť a stabilitu konvergencie
2. **počet neurónov a dropout miera**: určujú kapacitu a regularizáciu modelu
3. **počet vrstiev a aktivačná funkcia**: ovplyvňujú nelinearitu a hĺbku reprezentácie

Aj tentokrát sme na ladenie použili knižnicu `optuna`. Pre prvú skupinu sme vybrali nasledovné rozsahy hodnôt. Veľkosť dávky sme špecifikovali na hodnoty 64, 128, 256 a 512. Rýchlosť učenia sme definovali od 0,00001 do 0,01, pričom sme použili logaritmickú mieru kroku pre efektívne pokrytie viacerých rádov hodnôt. Vykonaných bolo 15 testov s použitím 5-násobnej krížovej validácie. Použitá konfigurácia je v tabuľke zaznačená v podobe `veľkosť dávky/rýchlosť učenia` a testy boli zoradené vzostupne podľa hodnôt parametrov. Výsledky testov sa nachádzajú v tabuľke 13.

Testy ukázali dôležitosť týchto dvoch parametrov, keďže v závislosti od ich konfigurácie sa výsledné F1-skóre výrazne menilo a rozdiel medzi najlepším a najhorším skóre bol až 0,172. Najvyššie skóre $0{,}681 \pm 0{,}013$ sme dosiahli pri konfigurácii s veľkosťou dávky 64 a faktorom rýchlosti učenia 0,00259. Okrem tejto konfigurácie sa viacero ďalších kombinácií priblížilo k podobnému výkonu, menovite `64/0,00203` ($0{,}680 \pm 0{,}012$), `64/0,00158` ($0{,}678 \pm 0{,}012$) a `64/0,00243` ($0{,}678 \pm 0{,}009$).

Tieto výsledky naznačujú, že optimálne hodnoty rýchlosti učenia sa pre model nachádzajú približne v intervale 0,0015 až 0,00226 a dávka s veľkosťou 64 je najefektívnejšia. Pri výrazne nižších alebo vyšších hodnotách rýchlosti učenia už dochádza k zhoršeniu výsledkov.

Z hľadiska veľkosti dávky tiež možno pozorovať, že jej zvyšovanie nevedie k zlepšeniu výkonu. Naopak, pri dávke 256 a rýchlosti učenia 0,00002 sme zaznamenali najhorší výsledok s F1-skóre len $0{,}509 \pm 0{,}019$. Aj pri väčších veľkostiach dávok sa však dali dosiahnuť relatívne dobré výsledky, pokiaľ bola rýchlosť učenia nastavená vhodne, napr. konfigurácia `256/0,00039` ($0{,}647 \pm 0{,}017$) alebo `512/0,00107` ($0{,}661 \pm 0{,}015$). Aj tak však zostávajú za najlepšími konfiguráciami s dávkou 64.

V druhej fáze sme ladili hyperparametre určujúce počet neurónov v skrytých vrstvách a dropout mieru. Pre počet neurónov sme na testovanie zvolili hodnoty 32, 64, 128,

Tabuľka 13: Výsledky ladenia hyperparametrov veľkosť dávky a rýchlosť učenia pre model MLP

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
64/0,00008	$0{,}739 \pm 0{,}009$	$0{,}653 \pm 0{,}008$	$0{,}613 \pm 0{,}024$	$0{,}623 \pm 0{,}019$
64/0,00010	$0{,}746 \pm 0{,}008$	$0{,}657 \pm 0{,}006$	$0{,}631 \pm 0{,}020$	$0{,}637 \pm 0{,}015$
64/0,00024	$0{,}754 \pm 0{,}006$	$0{,}667 \pm 0{,}011$	$0{,}648 \pm 0{,}015$	$0{,}652 \pm 0{,}013$
64/0,00158	$0{,}769 \pm 0{,}007$	$0{,}692 \pm 0{,}008$	$0{,}675 \pm 0{,}014$	$0{,}678 \pm 0{,}012$
64/0,00203	$0{,}769 \pm 0{,}006$	$0{,}691 \pm 0{,}009$	$0{,}678 \pm 0{,}014$	$0{,}680 \pm 0{,}012$
64/0,00243	$0{,}769 \pm 0{,}004$	$0{,}691 \pm 0{,}009$	$0{,}676 \pm 0{,}013$	$0{,}678 \pm 0{,}009$
64/0,00259	$0{,}773 \pm 0{,}006$	$0{,}698 \pm 0{,}013$	$0{,}672 \pm 0{,}015$	$0{,}681 \pm 0{,}013$
64/0,00770	$0{,}767 \pm 0{,}008$	$0{,}697 \pm 0{,}010$	$0{,}665 \pm 0{,}015$	$0{,}675 \pm 0{,}012$
64/0,00928	$0{,}770 \pm 0{,}006$	$0{,}693 \pm 0{,}012$	$0{,}667 \pm 0{,}015$	$0{,}676 \pm 0{,}010$
128/0,00132	$0{,}765 \pm 0{,}004$	$0{,}686 \pm 0{,}008$	$0{,}668 \pm 0{,}015$	$0{,}672 \pm 0{,}012$
128/0,00918	$0{,}766 \pm 0{,}010$	$0{,}689 \pm 0{,}012$	$0{,}664 \pm 0{,}025$	$0{,}672 \pm 0{,}020$
256/0,00002	$0{,}690 \pm 0{,}008$	$0{,}596 \pm 0{,}013$	$0{,}501 \pm 0{,}014$	$0{,}509 \pm 0{,}019$
256/0,00039	$0{,}751 \pm 0{,}007$	$0{,}665 \pm 0{,}010$	$0{,}641 \pm 0{,}020$	$0{,}647 \pm 0{,}017$
512/0,00006	$0{,}710 \pm 0{,}016$	$0{,}611 \pm 0{,}022$	$0{,}544 \pm 0{,}035$	$0{,}556 \pm 0{,}040$
512/0,00107	$0{,}759 \pm 0{,}007$	$0{,}675 \pm 0{,}014$	$0{,}658 \pm 0{,}016$	$0{,}661 \pm 0{,}015$

256 a 512. Interval dropout miery sme definovali od 0,1 do 0,7 s krokom 0,1. Celkovo sme vykonali 13 experimentov, každý s 5-násobnou krížovou validáciou. Výsledky sú v tabuľke 14, kde boli zoradené vzostupne podľa hodnôt parametrov. Tie sa v tabuľke uvádzajú v tvare `počet neurónov/dropout miera`.

Najlepší výsledok s F1-skóre $0{,}683 \pm 0{,}012$ dosiahla konfigurácia s 512 neurónmi a dropout mierou 0,2. Veľmi podobné výsledky dosiahli aj konfigurácie `256/0,2` ($0{,}682 \pm 0{,}006$) a `256/0,3` ($0{,}681 \pm 0{,}009$). To naznačuje, že optimálny počet neurónov v skrytej vrstve sa pohybuje medzi 256 a 512 neurónmi spolu s dropout mierou medzi 0,2 a 0,3.

Menší počet neurónov nedokázal dostatočne reprezentovať komplexitu dát, nezávisle od použitej dropout miery, čo sa prejavilo aj na horších výsledkoch. Výnimkou je však relatívne jednoduchá konfigurácia `64/0,1`, ktorá dosiahla solídny výsledok v podobe F1-skóre $0{,}674 \pm 0{,}009$. Mohla by byť vhodnou voľbou v prípadoch, kde je vyžadovaná nižšia výpočtová náročnosť.

Výsledky tiež ukazujú, že okrem počtu neurónov je dôležitá aj správne zvolená dropout miera, príliš nízka alebo naopak príliš vysoká hodnota má veľký vplyv na dosiahnutý výsledok. Túto skutočnosť je možné najviac pozorovať medzi `64/0,1` a

Tabuľka 14: Výsledky ladenia hyperparametrov počet neurónov a dropout miera pre model MLP

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
`32/0,4`	$0{,}752 \pm 0{,}007$	$0{,}672 \pm 0{,}012$	$0{,}639 \pm 0{,}014$	$0{,}647 \pm 0{,}013$
`32/0,5`	$0{,}755 \pm 0{,}007$	$0{,}683 \pm 0{,}017$	$0{,}621 \pm 0{,}011$	$0{,}641 \pm 0{,}012$
`64/0,1`	$0{,}767 \pm 0{,}004$	$0{,}684 \pm 0{,}010$	$0{,}672 \pm 0{,}011$	$0{,}674 \pm 0{,}009$
`64/0,3`	$0{,}766 \pm 0{,}006$	$0{,}683 \pm 0{,}012$	$0{,}658 \pm 0{,}014$	$0{,}665 \pm 0{,}009$
`128/0,5`	$0{,}766 \pm 0{,}009$	$0{,}689 \pm 0{,}013$	$0{,}662 \pm 0{,}015$	$0{,}669 \pm 0{,}014$
`128/0,7`	$0{,}755 \pm 0{,}005$	$0{,}678 \pm 0{,}007$	$0{,}638 \pm 0{,}010$	$0{,}650 \pm 0{,}008$
`256/0,1`	$0{,}768 \pm 0{,}008$	$0{,}692 \pm 0{,}011$	$0{,}669 \pm 0{,}024$	$0{,}676 \pm 0{,}016$
`256/0,2`	$0{,}770 \pm 0{,}006$	$0{,}695 \pm 0{,}007$	$0{,}677 \pm 0{,}008$	$0{,}682 \pm 0{,}006$
`256/0,3`	$0{,}771 \pm 0{,}005$	$0{,}696 \pm 0{,}008$	$0{,}676 \pm 0{,}010$	$0{,}681 \pm 0{,}009$
`256/0,5`	$0{,}767 \pm 0{,}008$	$0{,}693 \pm 0{,}013$	$0{,}669 \pm 0{,}019$	$0{,}674 \pm 0{,}017$
`512/0,1`	$0{,}768 \pm 0{,}005$	$0{,}686 \pm 0{,}008$	$0{,}677 \pm 0{,}012$	$0{,}678 \pm 0{,}008$
`512/0,2`	$0{,}772 \pm 0{,}004$	$0{,}695 \pm 0{,}009$	$0{,}680 \pm 0{,}014$	$0{,}683 \pm 0{,}012$
`512/0,7`	$0{,}769 \pm 0{,}010$	$0{,}694 \pm 0{,}013$	$0{,}673 \pm 0{,}021$	$0{,}678 \pm 0{,}018$

`64/0,3`, kde vyšší dropout zhoršil skóre o 0,009 bodu a medzi `128/0,5` a `128/0,7`, kde došlo k zhoršeniu až o 0,019 bodu.

Napriek tomu, že najvyššie skóre dosiahla konfigurácia `512/0,2`, rozhodli sme sa ďalej pokračovať s `256/0,2`. Dôvodom je najmä značné zvýšenie výpočtových nárokov, za cenu len marginálneho zlepšenia. Konfigurácia `256/0,2` má takisto štandardnú odchýlku len 0,006 bodu, čo je o polovicu menej než pre `512/0,2`.

V tretej fáze ladenia hyperparametrov sme testovali vplyv počtu vrstiev a použitej aktivačnej funkcie. Pod pojmom „vrstva“ v tomto kontexte rozumieme sekvenciu jednej plne prepojenej vrstvy, nasledovanú aktivačnou funkciou a ukončenú dropout vrstvou. Počet vrstiev teda určuje počet takýchto po sebe idúcich blokov. Aby sme znížili potrebný počet testov, rozhodli sme sa pre každú vrstvu použiť rovnaké parametre, čiže 256 neurónov a dropout mieru 0,2. Aktivačné funkcie zahrnuté do experimentov boli ReLU, Leaky ReLU a ELU, pričom každá bola testovaná s 1, 2 a 3 vrstvami, čím vzniklo celkom 9 možných konfigurácií. Pri testovaní bola použitá 5-násobná krížová validácia. Výsledky sú v tabuľke 15 a konfigurácia v nich je uvedená v tvare `počet vrstiev/aktivačná funkcia`.

Tabuľka 15: Výsledky ladenia hyperparametrov počet vrstiev a aktivačná funkcia pre model MLP

Konfigurácia	Správnosť	Presnosť	Senzitivita	F1-skóre
`1/ReLU`	$0{,}770 \pm 0{,}006$	$0{,}695 \pm 0{,}007$	$0{,}677 \pm 0{,}008$	$0{,}682 \pm 0{,}006$
`2/ReLU`	$0{,}769 \pm 0{,}006$	$0{,}689 \pm 0{,}013$	$0{,}668 \pm 0{,}010$	$0{,}674 \pm 0{,}009$
`3/ReLU`	$0{,}773 \pm 0{,}004$	$0{,}704 \pm 0{,}008$	$0{,}666 \pm 0{,}016$	$0{,}679 \pm 0{,}010$
`1/Leaky ReLU`	$0{,}767 \pm 0{,}008$	$0{,}683 \pm 0{,}016$	$0{,}677 \pm 0{,}010$	$0{,}676 \pm 0{,}012$
`2/Leaky ReLU`	$0{,}772 \pm 0{,}004$	$0{,}696 \pm 0{,}011$	$0{,}673 \pm 0{,}010$	$0{,}678 \pm 0{,}008$
`3/Leaky ReLU`	$0{,}769 \pm 0{,}004$	$0{,}690 \pm 0{,}013$	$0{,}672 \pm 0{,}014$	$0{,}676 \pm 0{,}011$
`1/ELU`	$0{,}749 \pm 0{,}006$	$0{,}668 \pm 0{,}010$	$0{,}634 \pm 0{,}018$	$0{,}646 \pm 0{,}013$
`2/ELU`	$0{,}757 \pm 0{,}005$	$0{,}675 \pm 0{,}010$	$0{,}656 \pm 0{,}010$	$0{,}658 \pm 0{,}011$
`3/ELU`	$0{,}758 \pm 0{,}003$	$0{,}676 \pm 0{,}008$	$0{,}651 \pm 0{,}006$	$0{,}655 \pm 0{,}003$

Najvyššie F1-skóre, $0{,}682 \pm 0{,}006$ sme dosiahli pri pôvodnej konfigurácii s 1 vrstvou a aktiváciou ReLU. Za ním nasledovali konfigurácie `3/ReLU` ($0{,}679 \pm 0{,}010$) a `2/Leaky ReLU` ($0{,}678 \pm 0{,}008$).

Z výsledkov je zrejmé, že ReLU je pre náš model najvhodnejšou aktivačnou funkciou a to najmä v kombinácii s jednou skrytou vrstvou. Leaky ReLU však takisto dosahuje dobré výsledky a pre konfiguráciu s dvoma vrstvami dokonca mierne prekonala ekvivalentnú ReLU verziu. Naopak, ELU dosahovala výrazne horšie výsledky naprieč všetkými tromi testovanými variantami. Jej najlepšia konfigurácia `2/ELU` dosiahla F1-skóre len $0{,}658 \pm 0{,}011$, čo je až o 0,024 bodu menej než najlepšia konfigurácia s ReLU.

Pokiaľ ide o počet vrstiev, výsledky ukazujú, že zvyšovanie počtu vrstiev neprinieslo výrazné zlepšenie výkonu (okrem ELU). Vo viacerých prípadoch sa dokonca objavil opačný efekt a došlo k zníženiu generalizačnej schopnosti modelu. To naznačuje, že pre naše dáta postačuje aj jednoduchšia architektúra siete, ktorá dosahuje nielen najvyššie F1-skóre, ale má aj nižšie výpočtové nároky než hlbšie siete.

Po dokončení optimalizácie tretej skupiny môžeme proces ladenia hyperparametrov považovať za ukončený. Najlepšie výsledky dosiahol model s veľkosťou dávky 64 a rýchlosťou učenia 0,00259, pričom tieto parametre sa ukázali ako najvplyvnejšie. Z architektonických parametrov sa najviac osvedčila konfigurácia s jednou skrytou vrstvou, 256 neurónmi, dropout mierou 0,2 a ReLU aktivačnou funkciou.

Pre túto finálnu konfiguráciu hyperparametrov sme na záver vykonali ešte 10-násobnú krížovú validáciu, ktorá priniesla F1-skóre $0{,}687 \pm 0{,}013$. V porovnaní s referenčnou

hodnotou $0{,}668 \pm 0{,}014$ ide o nárast o takmer 0,02 bodu, čo predstavuje významné zlepšenie detekčnej schopnosti modelu.

5.6 Porovnanie a vyhodnotenie klasifikátorov

Významnú časť tejto práce sme venovali implementácii a optimalizácii našich troch klasifikačných modelov. V tejto podkapitole si teraz zhrnieme dosiahnuté výsledky jednotlivých modelov a porovnáme ich medzi sebou. Následne bližšie analyzujeme správanie modelov podľa kategórií na základe konfúznych matíc.

Tabuľka 16: Výsledky finálnych klasifikačných modelov

Model	**Správnosť**	**Presnosť**	**Senzitivita**	**F1-skóre**
`RF`	$0{,}792 \pm 0{,}007$	$0{,}775 \pm 0{,}013$	$0{,}687 \pm 0{,}013$	$0{,}717 \pm 0{,}012$
`XGB`	$0{,}803 \pm 0{,}008$	$0{,}749 \pm 0{,}010$	$0{,}726 \pm 0{,}015$	$0{,}733 \pm 0{,}012$
`MLP`	$0{,}773 \pm 0{,}008$	$0{,}699 \pm 0{,}012$	$0{,}682 \pm 0{,}017$	$0{,}687 \pm 0{,}013$

V tabuľke 16 sú uvedené výsledky pre finálne modely s použitím 10-násobnej krížovej validácie. Najlepší celkový výkon spomedzi testovaných modelov dosiahol XGBoost. Vynikal v metrikách správnosti ($0{,}803 \pm 0{,}008$) a najmä v F1-skóre ($0{,}733 \pm 0{,}012$), čo poukazuje na dobrý pomer medzi presnosťou a senzitivitou. Vzhľadom na tieto výsledky sa XGBoost javí ako najvhodnejší model pre klasifikáciu kategórií malvéru. Nie je to prekvapením, keďže XGBoost sa v súčasnosti radí medzi najefektívnejšie algoritmy pre prácu s tabulárnymi dátami. Napriek tomu sme však očakávali o niečo lepšie výsledky, obzvlášť v porovnaní s náhodným lesom.

Tým sa dostávame k výsledkom modelu náhodného lesa. Dosiahol až prekvapivo podobné výsledky ako XGBoost. V metrikách správnosti ($0{,}792 \pm 0{,}007$) a F1-skóre ($0{,}717 \pm 0{,}012$) zaostával len mierne, a v metrike presnosti ($0{,}775 \pm 0{,}013$) ho dokonca aj prekonal. Jeho slabšou stránkou je však senzitivita ($0{,}687 \pm 0{,}013$), ktorá je podstatne nižšia, čo vysvetľuje aj nižšie celkové F1-skóre. Naznačuje to, že model má väčší problém identifikovať pozitívne prípady. Napriek tomu ide o dobrý klasifikátor, ktorý by mohol byť vhodný v prípadoch, kde je dôležitejšia presnosť ako senzitivita.

Viacvrstvový perceptrón sa v porovnaní s ostatnými modelmi ukázal ako najslabší. Dosiahol najnižšie hodnoty vo všetkých sledovaných metrikách, predovšetkým v F1-skóre ($0{,}687 \pm 0{,}013$), kde za modelom XGBoost zaostáva až o 0,046 bodu. Je zjavné, že model trpel nedostatočným počtom príznakov, ktorý sa ukázal už pri ladení parametrov vektorizácie (podkapitola 5.2.1). Napriek tomu sa nám optimalizáciou jeho hyperparametrov, na rozdiel od zvyšných modelov, podarilo dosiahnuť výrazné zlepšenie.

Je teda možné, že pri väčšom počte vstupných príznakov by bol schopný konkurovať alebo dokonca aj prekonať model XGBoost.

Celkovo považujeme dosiahnuté výsledky modelov za uspokojivé, keďže sa nám podarilo prekonať výsledky vzorových modelov pre dataset VirusShare a tým splniť cieľ definovaný na začiatku kapitoly 5. Náš model XGBoost dosiahol F1-skóre $0{,}733 \pm 0{,}012$, čím prekonal vzorový model XGBoost so skóre 0,718 a dosiahol tak zlepšenie o 0,015 bodu. Náš druhý model RF zaznamenal F1-skóre $0{,}717 \pm 0{,}012$, čím výrazne prekonal vzorový model RF so skóre iba 0,602. Ide teda o nárast až o 0,115 bodu, čo považujeme za zásadné zlepšenie.

5.6.1 Analýza výsledkov modelov podľa kategórii

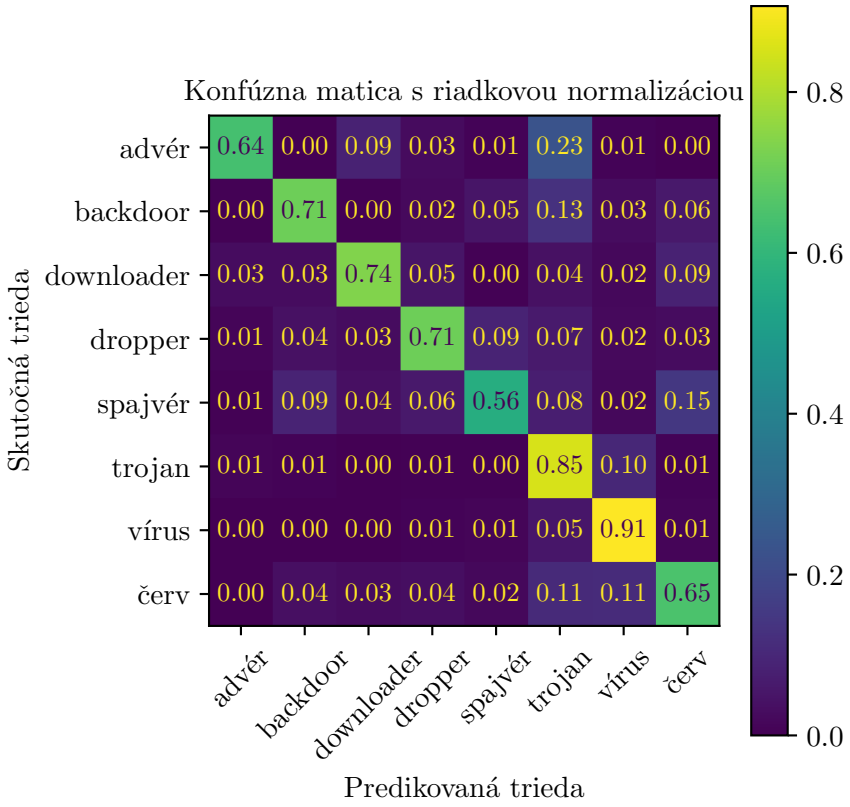
Aby sme lepšie porozumeli správaniu jednotlivých modelov pri klasifikácii, analyzovali sme ich výstupy aj pomocou konfúznych matíc. Tie nám umožňujú vidieť, ktoré kategórie boli najčastejšie zamenené a kde modely robili najviac chýb. Pre každý model sme vytvorili dve verzie konfúznych matíc. Prvým typom sú riadkovo normalizované matice, ktoré zobrazujú rozdelenie predikcií v rámci skutočnej triedy. Zjednodušene, pýtame sa „ak je skutočná trieda X, ako často ju model správne predpovedal?“. Tento pohľad zodpovedá metrike senzitivity.

Druhým typom sú stĺpcovo normalizované matice, ktoré naopak ukazujú, aké percento prípadov bolo správne klasifikovaných v rámci každej predikovanej triedy. Tu sa vlastne pýtame „ak model predpovedal triedu X, ako často to skutočne bola trieda X?“. Analogicky, tento pohľad zodpovedá metrike presnosti.

Na obrázku 2 je zobrazená konfúzna matica s riadkovou normalizáciou pre model XGBoost. Vyplýva z neho, že model dosahuje najvyššiu senzitivitu pri triedach vírus (0,91) a trojan (0,85). Pomerne dobré výsledky dosahuje aj pre downloader (0,74), dropper (0,71) a backdoor (0,71).

Najväčšie chyby boli zaznamenané pre triedu spajvér, kde bolo správne klasifikovaných len 56% prípadov. Výrazný podiel vzoriek tejto triedy bol nesprávne zaradený do tried červ (0,15), backdoor (0,09) a trojan (0,08). Pravdepodobnou príčinou je nízky počet trénovacích vzoriek pre spajvér, keďže ide o najmenej zastúpenú triedu v dátach, čo sťažuje modelu jej spoľahlivé rozpoznanie.

Pomerne málo správne identifikovaných vzoriek mala aj trieda advér (0,64). Zaujímavé je však, že až 23% vzoriek tejto triedy bolo nesprávne označených ako trojan, čo je najvýraznejšia zaujatosť voči konkrétnej triede naprieč celou maticou. Len o trochu lepšiu senzitivitu mala trieda červ (0,65), ktorá bola pomerne často zamieňaná s vírusom (0,11) a s trojanom (0,11).

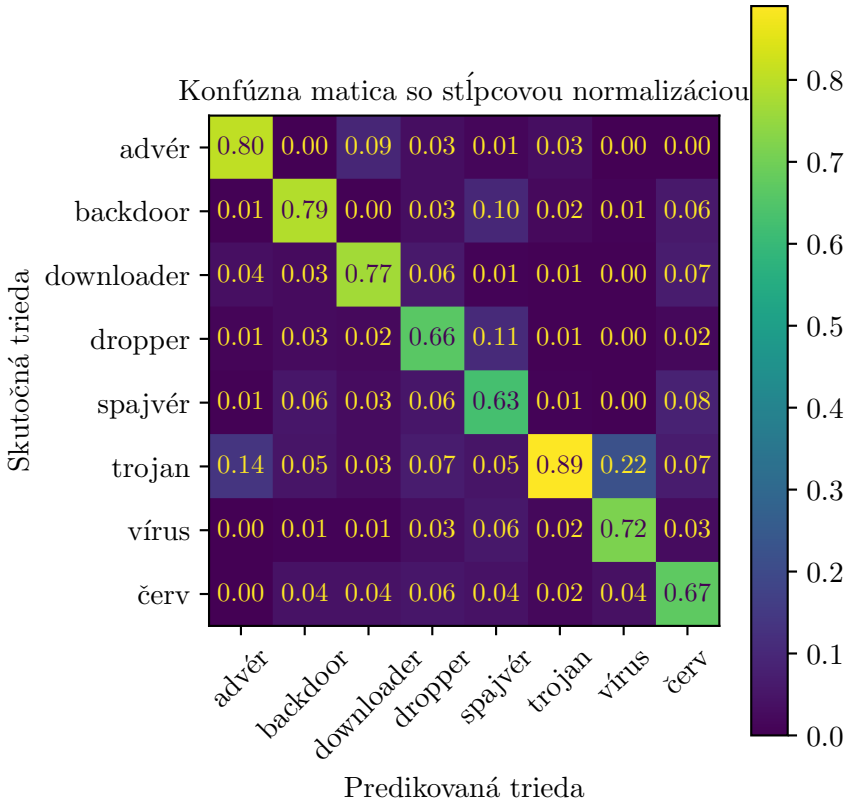


Obr. 2: Konfúzna matica pre model XGBoost s riadkovou normalizáciou

Trieda trojan bola celkovo najčastejšie nesprávne predikovanou triedou. Objavuje sa ako majoritný chybný výstup vo väčšine ostatných tried. Najpravdepodobnejším dôvodom je nevyváženosť dát v jeho prospech. Možnosťou sú však aj spoločné charakteristiky správania s inými typmi malvéru.

Konfúzna matica so stĺpcovou normalizáciou pre model XGBoost sa nachádza na obrázku 3. Model dosahuje najvyššiu presnosť pre triedy trojan (0,89) a advér (0,80), relatívne spoľahlivo identifikuje tiež backdoor (0,79), downloader (0,77) a vírus (0,72). Tu je však zaujímavé, že model má pomerne veľký problém rozlíšiť triedu vírus od trojan a robí chybnú predikciu až v 22% prípadoch.

Najnižšiu presnosť sme zaznamenali pre triedy spajvér (0,63) a dropper (0,66). Spajvér je najčastejšie zamieňaný za dropper (0,11) a backdoor (0,10). Dropper má chybové predikcie pomerne vyrovnane rozložené naprieč všetkými triedami od 0,03 až po 0,07. Táto skutočnosť však nie je prekvapivá, keďže obe triedy sú najmenej početné v datasete. Model preto nemá dostatok dát, aby sa ich naučil spoľahlivo rozlišovať. To sa však nedá povedať o triede červ, ktorá aj tak dosiahla presnosť len 67%. Tá bola najčastejšie mylne označovaná ako spajvér (0,08), trojan (0,07), downloader (0,07) a backdoor (0,06).

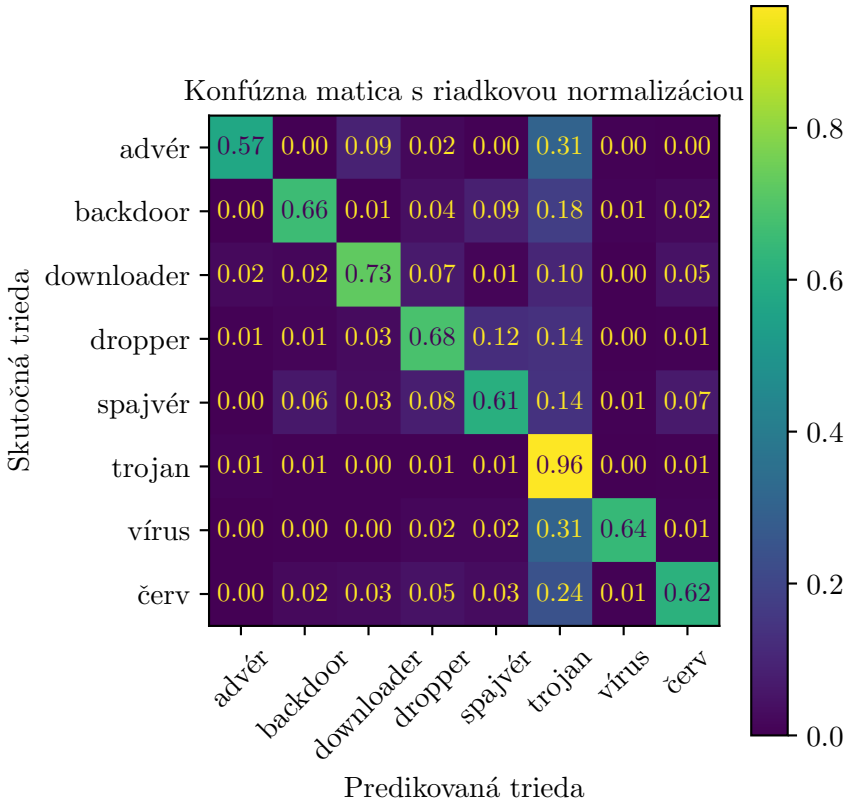


Obr. 3: Konfúzna matica pre model XGBoost so stĺpcovou normalizáciou

Na obrázku 4 sa nachádza konfúzna matica pre model náhodného lesa s riadkovou normalizáciou. Model dosahuje veľmi vysokú senzitivitu pre triedu trojan, kde bolo správne klasifikovaných až 96% prípadov. V prípade ostatných tried je však senzitivita výrazne nižšia. Druhou najúspešnejšou triedou je downloader so senzitivitou 0,73, čo predstavuje rozdiel až 23% oproti trojanu. Zvyšné triedy sa pohybujú v podobnom rozsahu, medzi 0,61 a 0,68. Výnimkou je trieda advér, kde model správne identifikoval iba 57% vzoriek, čo je zároveň aj najhorší výsledok.

Z konfúznej matice je zrejmé, že model má tendenciu výrazne nadhodnocovať výskyt triedy trojan, keďže táto predikcia sa vyskytuje ako najčastejšia chyba vo všetkých ostatných triedach. Najviac nesprávne zaradených do trojanu boli vzorky z tried advér (0,31), vírus (0,31), červ (0,24) a backdoor (0,18).

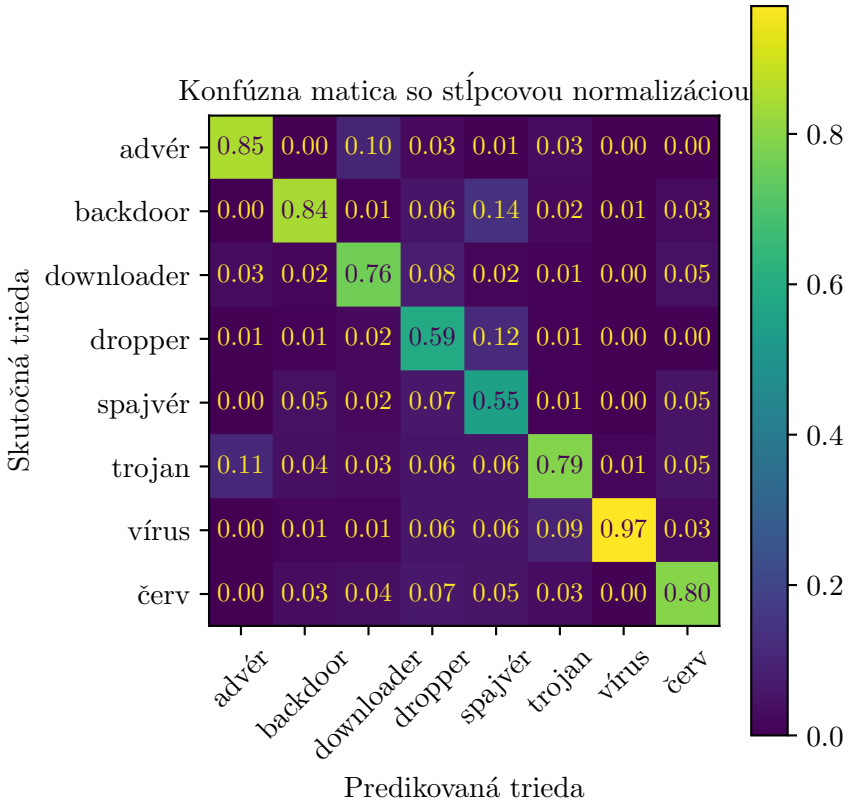
Vzhľadom na to, že bol použitý algoritmus náhodného lesa špeciálne navrhnutý pre nevyvážené datasety, by sme očakávali vyrovnanejšiu senzitivitu naprieč všetkými triedami. Výsledky však ukazujú, že značná zaujatosť v prospech trojanu pretrváva. Jediným výraznejším zlepšením oproti modelu XGBoost, ktorý bol trénovaný na nevyváženom datasete, je trieda spajvér. Tam model dosiahol senzitivitu 0,61 (oproti 0,56 pre XGBoost).



Obr. 4: Konfúzna matica pre model RF s riadkovou normalizáciou

Na obrázku 5 je znázornená stĺpcovo normalizovaná konfúzna matica pre model náhodného lesa. Veľmi vysokú presnosť dosahuje model pre triedu vírus (0,97) a takisto pre triedy advér (0,85), backdoor (0,84) a červ (0,80). Uspokojivú presnosť dosahuje aj pre trojan (0,79) a downloader (0,76). Stojí za zmienku, že trojan sa podľa presnosti umiestnil až na piatom mieste. V tomto prípade zrejme zafungoval vyvažovací algoritmus. Zlepšenie presnosti oproti modelu XGBoost nastalo pri štyroch triedach, menovite advér a backdoor zaznamenali nárast o 0,05 bodu, vírus o 0,10 bodu a červ až o 0,13 bodu.

Na druhej strane, aj napriek vyvažovaniu, pre triedy spajvér (0,55) a dropper (0,59) sme zaznamenali veľmi nízku presnosť. Dokonca je ešte nižšia než pri nevyváženom modeli XGBoost, konkrétne pokles o 0,08 bodu pre spajvér a 0,07 bodu pre dropper. Navyše, tieto dve triedy sú medzi sebou často zamieňané. Až 12% prípadov spajvéru bolo nesprávne klasifikovaných ako dropper, zatiaľ čo 7% dropper prípadov bolo označených ako spajvér. Trieda spajvér bola okrem toho často nesprávne identifikovaná aj ako backdoor (0,14), trojan (0,06) a vírus (0,06). Podobne aj dropper bol často zamieňaný za triedu downloader (0,08) a červ (0,07). Napriek tomu, že sú dáta vyvážené, tak jednotlivé stromy opakovane vidia len obmedzený počet vzoriek. To môže byť vysvetlením nedostatočnej generalizácie na validačnú množinu. Ďalším faktorom môže byť aj



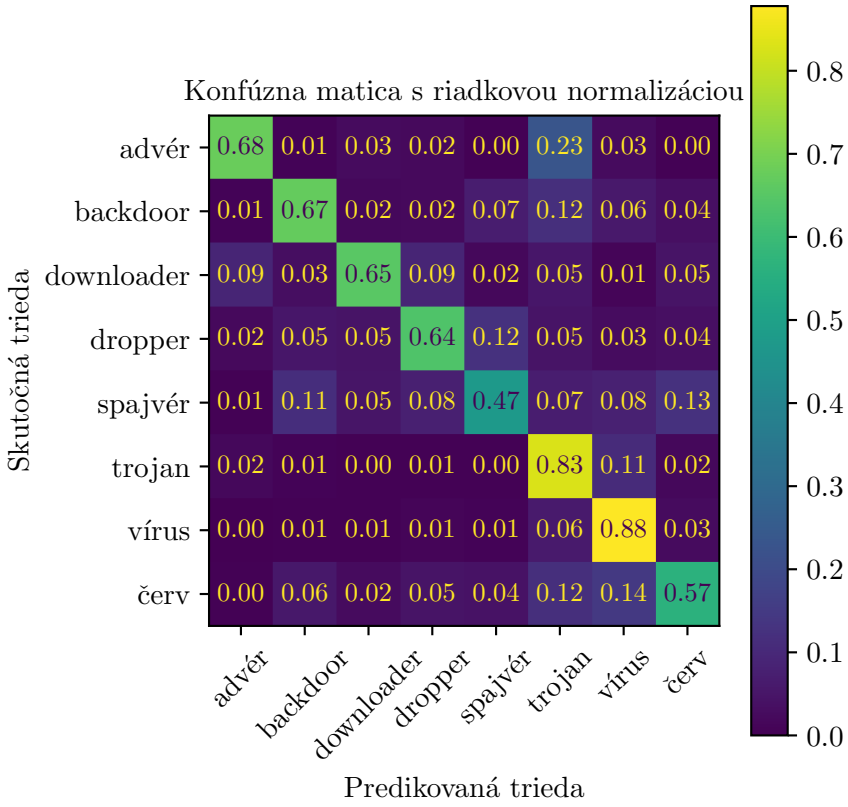
Obr. 5: Konfúzna matica pre model RF so stĺpcovou normalizáciou

podobnosť medzi triedami, ktoré sú najčastejšie zamieňané, to však zatiaľ nemáme ako overiť.

Na obrázku 6 je zobrazená konfúzna matica pre model MLP s riadkovou normalizáciou. Model dosahuje najvyššiu senzitivitu pre triedy vírus (0,88) a trojan (0,83). Ide o veľmi dobré výsledky, len o niečo nižšie než v prípade modelu XGBoost (vírus -0,003 a trojan -0,002). Najbližšia ďalšia trieda podľa senzitivity je však až advér s hodnotou 0,68, čo predstavuje pokles o 15%. Nasledujú triedy backdoor (0,67), downloader (0,65) a dropper (0,64), ktoré dosahujú pomerne vyrovnané výsledky.

Najhorší výsledok bol zaznamenaný pre triedu spajvér, ktorú model správne klasifikoval len v 47% prípadoch. Ide o najslabší výsledok naprieč všetkými modelmi. Najčastejšie bol spajvér nesprávne klasifikovaný ako triedy červ (0,13) a backdoor (0,11). Slabý výsledok model dosiahol aj pre triedu červ s hodnotou senzitivity len 0,57. Tá bola veľmi často zamieňaná za triedy vírus (0,14) a trojan (0,12).

Všimnúť si je takisto možné, že pre triedu advér sú takmer všetky mylné predikcie smerované na triedu trojan (0,23). Napriek tomu, že trieda trojan sa vyskytuje pomerne často medzi chybnými predikciami vo všetkých modeloch, vzorky pre advér sú do nej priraďované oveľa frekventovanejšie. Tento jav sa nevyskytuje len pri MLP, rovnaký trend



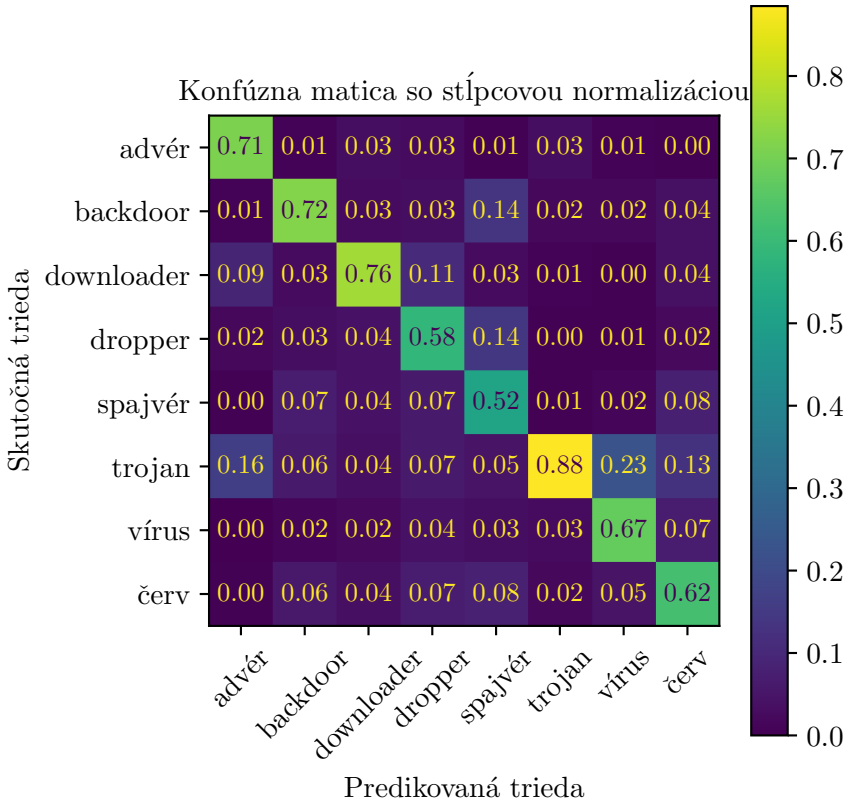
Obr. 6: Konfúzna matica pre model MLP s riadkovou normalizáciou

pozorujeme aj pri modeloch XGBoost (0,23) a RF (0,31). To napovedá, že vysvetlením by mohli byť veľmi podobné sekvencie API volaní vyskytujúce sa vo vzorkách pre obe triedy.

Na obrázku 7 je zobrazená konfúzna matica pre model MLP so stĺpcovou normalizáciou. Model vykazuje najvyššiu presnosť pre triedu trojan s hodnotou 0,88. Za ňou s veľkým poklesom, ale stále pomerne dobrou presnosťou, nasledujú triedy downloader (0,76), backdoor (0,72) a advér (0,71).

Chybné predikcie triedy advér boli najčastejšie urobené pre skutočné vzorky trojanu (0,16). Opäť, tento jav vieme pozorovať naprieč všetkými modelmi a predpokladáme, že súvisí s rovnakými sekvenciami API volaní pre obe triedy.

Podobnú skúsenosť máme aj s triedou vírus. MLP model ju predpovedal s 67% presnosťou, no najviac mylných predikcií urobil pre vzorky triedy trojan (0,23). Takmer totožné percento chyby sa vyskytlo aj na modeli XGBoost (0,22). Avšak pre model náhodného lesa bola rovnaká chyba len zanedbateľná (0,01). To naznačuje, že nie je dôsledkom podobných príznakov, ale skôr nedostatočného počtu vzoriek pre triedu vírus.



Obr. 7: Konfúzna matica pre model MLP so stĺpcovou normalizáciou

Najhoršie výsledky presnosti model preukázal pri predikcií vzoriek pre triedy spajvér (0,52), dropper (0,58) a červ (0,62). Predikcia spajvéru bola najčastejšie mylná pre vzorky tried dropper (0,14) a backdoor (0,14). Podobný výsledok sme zaznamenali vo všetkých modeloch, predpokladáme teda, že sa v týchto vzorkách vyskytujú podobné príznaky.

Trieda dropper je síce najviac chybne predikovaná pre downloader (0,11), avšak zvyšné chyby sú rozložené naprieč ostatnými triedami. Totožný jav sa objavuje aj pri ostatných modeloch, čo ukazuje buď na nedostatočný počet reprezentatívnych vzoriek alebo nedostatočne špecifické príznaky.

6 Aplikácia vysvetľovacích metód

V tejto kapitole sa budeme venovať aplikácii vysvetľovacích metód na výstupy našich klasifikačných modelov. Na vysvetlenie sme pre každý model a každú triedu vybrali päť náhodných vzoriek z testovacej množiny. Uistili sme sa pritom, že vybrané vzorky model správne klasifikuje do skutočných tried. Následne sme na tieto vzorky aplikovali vysvetľovacie metódy SHAP, Anchors a kontrafaktuály. Celkovo sme tak mali k dispozícii až 360 rôznych vysvetlení výstupov. Kvôli značnému rozsahu sme vykonali analýzu dvoch vzoriek pre každý model. Všetky vysvetlenia sa však nachádzajú v adresároch, ako to je uvedené v prílohe A.

Hlavným cieľom tejto časti práce je porovnať výstupy z vysvetľovacích metód a vyvodiť z nich závery o použiteľnosti týchto metód na pochopenie správania klasifikačných modelov pre detekciu kategórií malvéru.

Na aplikáciu SHAP a kontrafaktuálnych metód bol použitý rovnaký systém ako sme uviedli v kapitole 5. Pre metódu Anchors sme, vzhľadom na jej vysoké výpočtové nároky, použili systém s procesorom Intel Core i9-10980XE (18 jadier), pamäťou RAM s kapacitou 256 GB a operačným systémom Debian 5.10.140-1.

V podkapitolách 6.1, 6.2 a 6.3 si stručne zhrnieme implementáciu a očakávané výstupy použitých vysvetľovacích metód. V podkapitolách 6.4, 6.5 a 6.6 sa následne budeme venovať aplikáciám vysvetľovacích metód na už konkrétne vybrané príklady pre jednotlivé modely.

6.1 Aplikácia SHAP v experimentoch

Na výpočet SHAP hodnôt sme použili Python knižnicu `shap`, ktorá poskytuje algoritmy prispôsobené konkrétnym typom modelov. Algoritmus TreeSHAP (podkapitola 3.2.2) sme použili pre XGBoost a jeho GPU verziu pre náhodný les a pre MLP zasa algoritmus DeepSHAP (3.2.3). Vďaka týmto špecializovaným prístupom sa nám podarilo značne zrýchliť a tiež zvýšiť presnosť výpočtu SHAP hodnôt.

Na vizualizáciu výstupov zo SHAP metód sme použili tzv. rozhodovacie grafy (angl. *decision plots*). Tento typ grafu umožňuje sledovať, ako sa predikcia modelu pre konkrétnu vzorku postupne mení v závislosti od príspevkov jednotlivých príznakov. Príznaky sú v grafe zoradené zostupne podľa veľkosti príspevku (najväčší je hore). Príspevky sú postupne odspodu sčítavané, čím vzniká smerovanie až k finálnemu rozhodnutiu modelu.

Každý z týchto príspevkov zodpovedá SHAP hodnote, ktorá vyjadruje, o koľko daný príznak posúva výstup modelu oproti základnej hodnote (angl. *base value*). Základná hodnota predstavuje očakávaný výstup modelu v prípade, že by žiadna z aktuálnych

hodnôt príznakov nebola známa. Výsledná predikcia sa potom rovná súčtu základnej hodnoty a príspevkov jednotlivých príznakov (podrobne sme SHAP rozoberali v podkapitole 3.2).

Treba ešte poznamenať, že príspevky sú na osi x reprezentované vo forme tzv. logitov (angl. *log-odds*). Ide o neupravené výstupy modelu ešte pred aplikáciou softmax funkcie, ktorá ich transformuje na pravdepodobnosti. Použitie logitov je výhodné najmä preto, že sú lineárne a aditívne. Vďaka tomu je možné presne sledovať, ako jednotlivé príznaky nezávisle prispievajú k výslednému skóre danej triedy.

Výnimkou je však model náhodného lesa, ktorý nevracia logity, ale priamo odhady pravdepodobností. Je to dané jeho interným fungovaním, keďže výstup vzniká ako priemer hlasovania jednotlivých stromov. Na rozhodovacom grafe sú preto na osi x zobrazené príspevky priamo vo forme pravdepodobností. Aj napriek tomu sú tieto príspevky, teda SHAP hodnoty, aditívne. Ich súčet spolu so základnou hodnotou zodpovedá finálnej predikovanej pravdepodobnosti. Tým, že sú ale v pravdepodobnostnom priestore, hodnoty blízke 0 alebo 1 nie sú plne lineárne, na čo treba pri interpretácii výsledkov dať pozor.

Pre lepšiu čitateľnosť sme v každom rozhodovacom grafe zobrazili len 15 najdôležitejších príznakov. Keďže grafy nezahŕňajú všetky príznaky (a tým ani všetky príspevky), počiatočný bod na osi x je už posunutý voči základnej hodnote (znázornenej šedou kolmou čiarou). Tento posun je očakávaný a nepredstavuje chybu v grafe. Hodnoty uvedené v zátvorkách zodpovedajú skutočným hodnotám príznakov pre danú vzorku.

6.2 Aplikácia Anchors v experimentoch

Na vytvorenie anchor pravidiel sme použili Python knižnicu `alibi`, ktorá rozširuje knižnicu `anchor-exp` od autora pôvodného článku. Pre každú z náhodne vybraných vzoriek sme hľadali dve verzie pravidiel, jednu s požadovanou presnosťou 0,9 a druhú s presnosťou 0,8. Zámerom bolo pozorovať zmeny v odhade pokrytia a počte predikátov v jednotlivých pravidlách (podkapitola 3.3).

Použitý algoritmus je extrémne výpočtovo náročný, obzvlášť pre veľký počet príznakov. V počiatočných fázach experimentov sme ho testovali aj na modeloch s väčším počtom príznakov než používajú naše finálne modely teraz. Konkrétne na modeloch s 10 000, 5 000 a 1 500 príznakmi. Každý z testovaných modelov však v priebehu výpočtu zlyhal, pretože jeho nároky na operačnú pamäť prekročili 256 GB, ktoré sme mali k dispozícii na použitom systéme. Empirickým testovaním sme určili, že praktický limit pre počet príznakov na danom systéme je približne 1 000.

Algoritmus tiež umožňuje nastaviť počet anchor kandidátov, ktoré si MAB v každej iterácii vyberie. Vyšší počet kandidátov znižuje riziko uviaznutia v lokálnom minime a zvyšuje kvalitu výsledného pravidla, no zároveň výrazne zvyšuje aj pamäťové nároky. Z tohto dôvodu sme boli nútení použiť len jedného kandidáta. Dôsledkom sú však vo viacerých prípadoch totožné výsledky pre obe hľadané verzie presnosti. Z výpočtových dôvodov sme tiež museli obmedziť maximálny počet podmienok v jednom anchor na 15. Vyššie hodnoty totiž znovu prekračovali kapacitu dostupnej pamäte. Toto obmedzenie však spôsobilo, že viacero anchors bolo predčasne ukončených ešte pred dosiahnutím požadovanej presnosti.

6.3 Aplikácia kontrafaktuálov v experimentoch

Na vytvorenie kontrafaktuálov sme opäť použili Python knižnicu `alibi`, konkrétne implementáciu založenú na prístupe CEGP, ktorý smeruje kontrafaktuály k prototypu cieľovej triedy (podkapitola 3.4). Cieľovú triedu sme pre náhodne vybrané vzorky žiadnym spôsobom nešpecifikovali, ale sme nechali algoritmus, aby si vybral najbližšiu triedu na základe k susedov.

Keďže ide o optimalizačný algoritmus, ktorý pracuje so stratovou funkciou, výsledné hodnoty nájdených kontrafaktuálnych príznakov sú vyjadrené ako reálne čísla. Takéto výstupy sú však pre naše potreby neprijateľné. Empirickým testovaním sme ale zistili, že ak nájdené hodnoty skonvertujeme do priestoru celých čísiel, tak výsledná kontrafaktuálna trieda a aj pravdepodobnosti tried zostanú zachované. Pre modely RF a MLP sme na túto konverziu použili zaokrúhľovanie na najbližšie celé číslo. V prípade modelu XGBoost však táto metóda nefungovala a namiesto toho bolo potrebné orezanie desatinnej časti čísla (napr. 1,99 sa upraví na 1). Tento rozdiel sme objavili empiricky a jeho príčina nám nie je zrejmá.

Upravené kontrafaktuálne hodnoty sme následne validovali na príslušných modeloch, pričom sme overili ich zhodu s výsledkami pôvodných kontrafaktuálov[18]. Finálnym výstupom našich vysvetlení sú práve tieto upravené a validované kontrafaktuály.

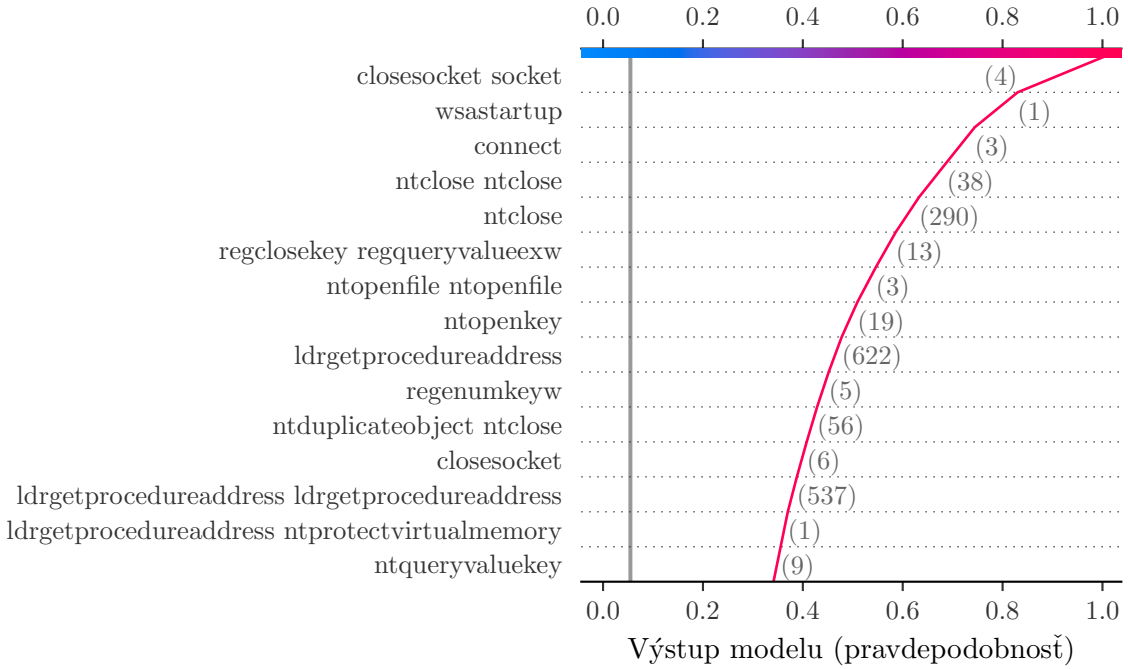
6.4 Vysvetlenia pre náhodný les

V tejto podkapitole analyzujeme vysvetlenia pre dve náhodne vybrané vzorky, ktoré boli správne klasifikované modelom náhodného lesa. Prvú z kategórie advér (vzorka 187) a druhú z kategórie backdoor (vzorka 170).

[18]Kontrola pravdepodobnosti tried nebola vykonaná exaktným porovnaním, ale pomocou metódy `numpy.allclose` s nastavenou relatívnou toleranciou 1E−5 a absolútnou toleranciou 1E−8. V prípade modelu MLP, pravdepodobne v dôsledku použitia `MinMaxScaler`, bolo potrebné zvýšiť absolútnu toleranciu na 0,1.

6.4.1 Vzorka 187

V tejto časti si urobíme analýzu pre vzorku 187 z triedy advér. Tri najdôležitejšie príznaky podľa SHAP sú `CloseSocket Socket`, `WSAStartup` a `Connect`. Naznačujú, že model si vytvoril silnú asociáciu s triedou advér a so sieťou spojenými aktivitami (obrázok 8).



Obr. 8: Rozhodovací graf SHAP pre vzorku 187 správne klasifikovanú modelom RF ako advér

`WSAStartup` je zodpovedné za inicializáciu knižnice `Winsock`, ktorá je nevyhnutná pre začatie nízkoúrovňovej sieťovej komunikácie (používanie soketov). `Socket` vytvára sieťový soket, čím v podstate nastavuje koncový bod pre výmenu dát. `Connect` nadviaže pripojenie na vzdialený server. `CloseSocket` je zodpovedné za ukončenie soketového pripojenia po dokončení komunikácie.

Takéto správanie je konzistentné s advérom. Môže naznačovať, že malvér nadväzuje krátkodobé spojenia s jedným alebo viacerými reklamnými servermi, ktoré využíva na aktualizáciu reklám, odosielanie telemetrických dát a podobne.

Okrem toho, prítomnosť veľkého počtu nízkoúrovňových systémových a registrových volaní (`LdrGetProcedureAddress`, `NtProtectVirtualMemory` a `RegQueryValueExW`) môže naznačovať použitie techník na utajenie a pretrvávanie v systéme. Najvýraznejším signálom modelu však zostáva vzor sieťovej aktivity.

Výstup z metódy Anchors (1) sa zhoduje so SHAP vysvetlením ohľadne dôležitosti sieťového spojenia a práce s registrami. Navyše, však určil nutnú prítomnosť sekvencie API volaní `GetSystemTimeAsFileTime GetFileAttributesW`, ktorá sa v rozhodovacom grafe neobjavila. Indikuje získanie systémového času a následnú kontrolu vlastnosti nejakého súboru. Môže naznačovať kontrolu, či nebeží v testovacom prostredí, alebo iné prispôsobenie správania podľa systému, v ktorom beží.

IF `RegOpenKeyExA RegQueryValueExA` > 0 **AND** `CloseSocket Socket` > 0
AND `GetSystemTimeAsFileTime GetFileAttributesW` > 0
THEN PREDICT Advér (1)

Presnosť: 0,9947 Pokrytie: 0,0014

Anchors si je týmto vysvetlím úplne istý, keďže odhadol presnosť až na 0,9947. Pokrytie je však veľmi nízke, čo naznačuje, že sa vzťahuje len na malú časť dát. Obe hľadané verzie presnosti našli rovnaký anchor.

Aplikácia zmien z kontrafaktuálnej metódy spôsobila preklopenie pôvodnej triedy advér (s pravdepodobnosťou 1,0) na triedu dropper s výslednou pravdepodobnosťou 0,285 a 0,18 pre advér. Zmeny potrebné na toto preklopenie sú uvedené v tabuľke 17 a výsledné pravdepodobnosti tried v tabuľke 18.

Tabuľka 17: Kontrafaktuálne zmeny pre vzorku 187 správne klasifikovanú modelom RF ako advér

	Hodnota	
Príznak	**Pôvodná**	**Zmenená**
`CloseSocket Socket`	4	2
`Connect`	3	1
`CopyFileA NtDelayExecution`	0	1
`LdrGetProcedureAddress NtProtectVirtualMemory`	1	0
`NtProtectVirtualMemory NtProtectVirtualMemory`	1	0
`WSAStartup`	1	0

Aj kontrafaktuálne vysvetlenie potvrdzuje naučenú asociáciu modelu pre triedu advér a sieťové aktivity. Zmeny totiž zahŕňajú zníženie počtu `CloseSocket Socket` zo 4 na 2, `Connect` z 3 na 1 a `WSAStartup` z 1 na 0. Treba si však uvedomiť, že tieto zmenené hodnoty nie sú úplne reálne, keďže bez zavolania `WSAStartup` by nemali byť vôbec prítomné ani zvyšné dve volania. Aj takéto výsledky sú však očakávané a môže

Tabuľka 18: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 187 správne klasifikovanú modelom RF ako advér

Vzorka	Advér	Backdoor	Downloader	Dropper	Spajvér	Trojan	Vírus	Červ
Pôvodná	**1,000**	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Zmenená	0,180	0,105	0,065	**0,285**	0,180	0,055	0,040	0,090

za ne spôsob, akým sú kontrafaktuály konštruované. Pokiaľ nie je manuálne určené obmedzenie na zmeny medzi určitými príznakmi, metóda hľadá aj medzi „nereálnymi“ hodnotami. Aj napriek tomu je však trend zrejmý.

Zmeny sa tiež ukázali v znížení počtu volaní pre `LdrGetProcedureAddress` a `NtProtectVirtualMemory` na 0. Ich dôležitosť potvrdil aj rozhodovací graf, avšak nezdá sa, že by im pripísal takú zásluhu na predikcii, akú majú podľa kontrafaktuálov.

Poslednou, no kľúčovou zmenou je tiež pribudnutie sekvencie volaní `CopyFileA` `NtDelayExecution`, ktorá bola zmenená z 0 na 1. Z toho je zrejmé, že model sa naučil správne rozpoznať, že kopírovanie súborov a následné pozastavenie vykonávania nie je typické pre advér. Zároveň nám to odhalilo, že toto správanie má najviac spojené s triedou dropper, čo zodpovedá jej charakteristike.

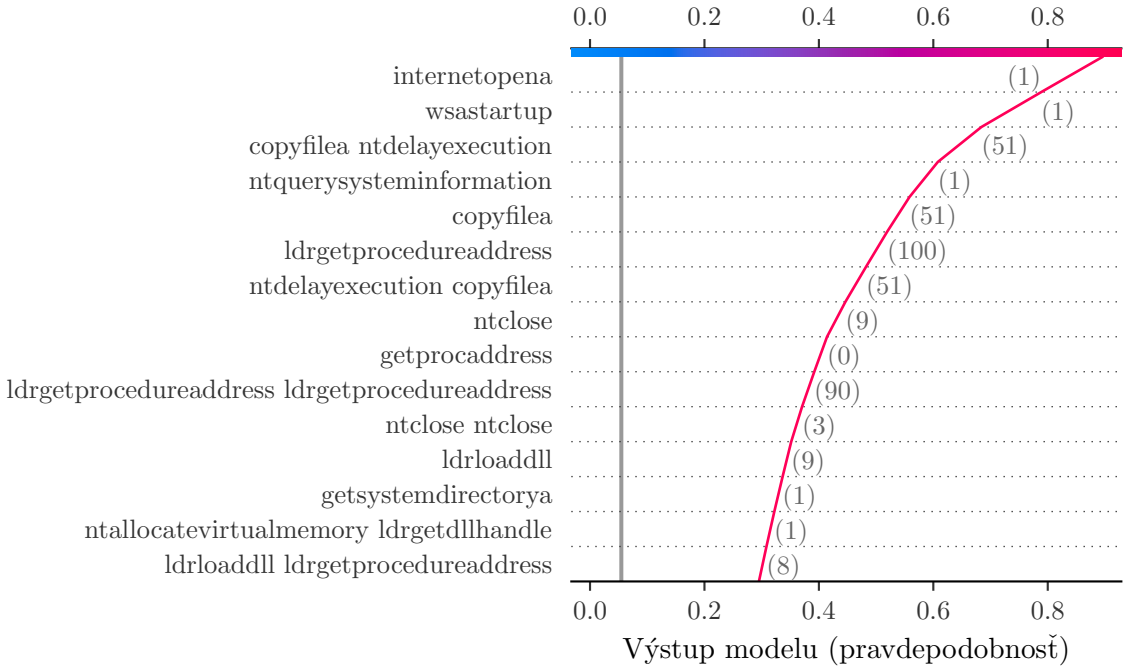
6.4.2 Vzorka 170

V tejto časti vykonáme analýzu vzorky 170, ktorá patrí do triedy backdoor. Na základe rozhodovacieho grafu (obrázok 9) model považuje za najvýznamnejšie indikátory výskyt volaní `InternetOpenA`, `WSAStartup` a `NtQuerySystemInformation`, ako aj vyšší počet volaní `CopyFileA` a `NtDelayExecution` vrátane ich n-gramov.

Backdoor potrebuje prístup na internet, aby umožnil útočníkovi vzdialený prístup k systému. API `InternetOpenA` slúži na inicializáciu relácie `WinINet`, ktorá umožňuje vykonávať vysokoúrovňové internetové operácie, ako napr. HTTP alebo FTP požiadavky. Model si teda správne spája volania potrebné pre prístup k internetu, `InternetOpenA` a `WSAStartup`, s triedou backdoor.

Čo však v grafe chýba, sú API vykonávajúce samotnú sieťovú komunikáciu ako napr. `Connect` či `InternetConnect`. Ide o podozrivé správanie, ktoré naznačuje používanie techník na ich utajenie, napr. pomocou dynamického vykonávania. Túto skutočnosť potvrdzuje v grafe prítomnosť až 100 volaní `LdrGetProcedureAddress`, ktoré sú často používané práve na takéto dynamické vykonávanie. Model sa teda správne naučil rozpoznávať aj takéto pokročilejšie techniky.

Za silný indikátor backdooru model tiež považuje prítomnosť príznakov `CopyFileA` `NtDelayExecution`, `CopyFileA` a `NtDelayExecution` `CopyFileA`. Možné si je tiež všim-



Obr. 9: Rozhodovací graf SHAP pre vzorku 170 správne klasifikovanú modelom RF ako backdoor

núť, že všetky tri sa nachádzajú v rovnakom počte 51, z čoho vyplýva, že ide o 51 sekvencií `NtDelayExecution CopyFileA NtDelayExecution`, ktoré boli rozdelené na n-gramy. Tento opakovaný vzorec naznačuje, že malvér sa kopírovanie súborov snaží časovo rozložiť, zrejme na zníženie rizika detekcie antivírusovým softvérom alebo pri analýze v sandbox prostredí. Tento predpoklad podporuje aj prítomnosť API `NtQuerySystemInformation`, ktoré môže backdoor použiť na detekciu prostredia a na základe toho upraviť časové oneskorenie.

Naproti tomu výstup z metódy Anchors (2), pre verziu s nižšou presnosťou, neurčil inicializáciu sieťového spojenia ako nutný predikát. Zhoduje sa však so SHAP vysvetlením v dôležitosti volaní `NtQuerySystemInformation` a `CopyFileA NtDelayExecution`. Okrem toho, za predikát určil aj sekvenciu `NtAllocateVirtualMemory LdrGetDllHandle`. Tá je síce prítomná v grafe, jej príspevok je však pomerne malý. Sekvencia vykonáva alokáciu pamäte, typicky ako prípravu pre vloženie *shellcode* do iného procesu, a následné dynamické načítanie DLL. To naznačuje pokus o obídenie detekcie, keďže sa vyhýba priamemu použitiu podozrivých API ako napr. `CreateRemoteThread` či `WriteProcessMemory`. Anchors udáva pomerne vysokú presnosť 0,8859, no pokrytie je iba 0,0049.

IF `NtQuerySystemInformation` > 0 **AND** `CopyFileA NtDelayExecution` > 0
AND `NtAllocateVirtualMemory LdrGetDllHandle` > 0
THEN PREDICT Backdoor

Presnosť: 0,8859 Pokrytie: 0,0049

(2)

Metóde Anchors sa podarilo nájsť aj pravidlo (3), ktoré spĺňa vyššiu požiadavku presnosti a to na hodnotu 0,9559 s pokrytím 0,0044. Toto navýšenie spôsobilo pribudnutie nového predikátu, ktorý určil, že vo vzorke sa nesmie nachádzať sekvencia `NtClose NtFreeVirtualMemory`. V kombinácii s predchádzajúcimi predikátmi takéto správanie naznačuje zámerné vyhýbanie uvoľneniu zdrojov, čo poukazuje na snahu o pretrvanie v pamäti.

IF `NtQuerySystemInformation` > 0 **AND** `CopyFileA NtDelayExecution` > 0
AND `NtAllocateVirtualMemory LdrGetDllHandle` > 0
AND `NtClose NtFreeVirtualMemory` $= 0$
THEN PREDICT Backdoor

Presnosť: 0,9559 Pokrytie: 0,0044

(3)

Analýza nájdených kontrafaktuálnych zmien (tabuľka 19) jasne ukázala, že model považuje za kľúčové API volania `InternetOpenA` a `WSAStartup`. V prípade, že ich hodnoty zmeníme na 0, tak vo výslednej pravdepodobnosti tried, backdoor klesne z pôvodnej hodnoty 0,895 na 0,016 a predikovaná trieda sa preklopí na červ s pravdepodobnosťou 0,348. Všetky pravdepodobnosti sú uvedené v tabuľke 20.

Tabuľka 19: Kontrafaktuálne zmeny pre vzorku 170 správne klasifikovanú modelom RF ako backdoor

	Hodnota	
Príznak	**Pôvodná**	**Zmenená**
`InternetOpenA`	1	0
`WSAStartup`	1	0

Môže sa zdať, že sme analýzou narazili na rozpor medzi vysvetleniami. Metódy SHAP a kontrafaktuály obe poukazujú na dôležitosť volaní `InternetOpenA` a `WSAStartup`. Keby však tieto príznaky vo vzorke neboli prítomné (mali by hodnotu 0), výsledné pravdepodobnosti sa na prvý pohľad nezhodujú. Ak si z rozhodovacieho grafu odmyslíme

Tabuľka 20: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 170 správne klasifikovanú modelom RF ako backdoor

Vzorka	Advér	Backdoor	Downloader	Dropper	Spajvér	Trojan	Vírus	Červ
Pôvodná	0,000	**0,895**	0,000	0,000	0,000	0,000	0,000	0,105
Zmenená	0,000	0,339	0,016	0,047	0,143	0,097	0,011	**0,348**

príspevky príslušných príznakov, pravdepodobnosť pre triedu backdoor by sa rovnala približne 0,7. Z kontrafaktuálneho vysvetlenia ale vyplýva, že skutočná pravdepodobnosť by v takom prípade bola len 0,339. Treba si však uvedomiť, že rozhodovací graf nič nehovorí o výstupe modelu v prípade iných hodnôt, než pre aké bol vytvorený. Konkrétne rozloženie príspevkov príznakov je teda zaručené iba pre jednu konkrétnu kombináciu príznakov.

Ďalším rozporom sa môže javiť, že Anchors nepovažuje príznaky `InternetOpenA` a `WSAStartup` za predikáty pre triedu backdoor. To však neznamená, že by model tieto príznaky ignoroval. Ako ukázali zvyšné dve metódy, ich prítomnosť výrazne zvyšuje pravdepodobnosť klasifikácie do triedy backdoor. Anchors sa však zameriava na hľadanie minimálnych podmienok, ktoré vedú ku konkrétnej predikcii. Výsledné pravidlo je potom *nezávislé* od hodnôt iných príznakov, čo sa o predchádzajúcich dvoch metódach nedá povedať. Tiež je ale nutné podotknúť, že Anchor je zo svojej podstaty stochastický, a teda môže existovať aj alternatívna kombinácia predikátov, ktorá zahŕňa spomínané príznaky. Ako sme však už vysvetlili (podkapitola 6.2), hľadali sme iba jedno pravidlo.

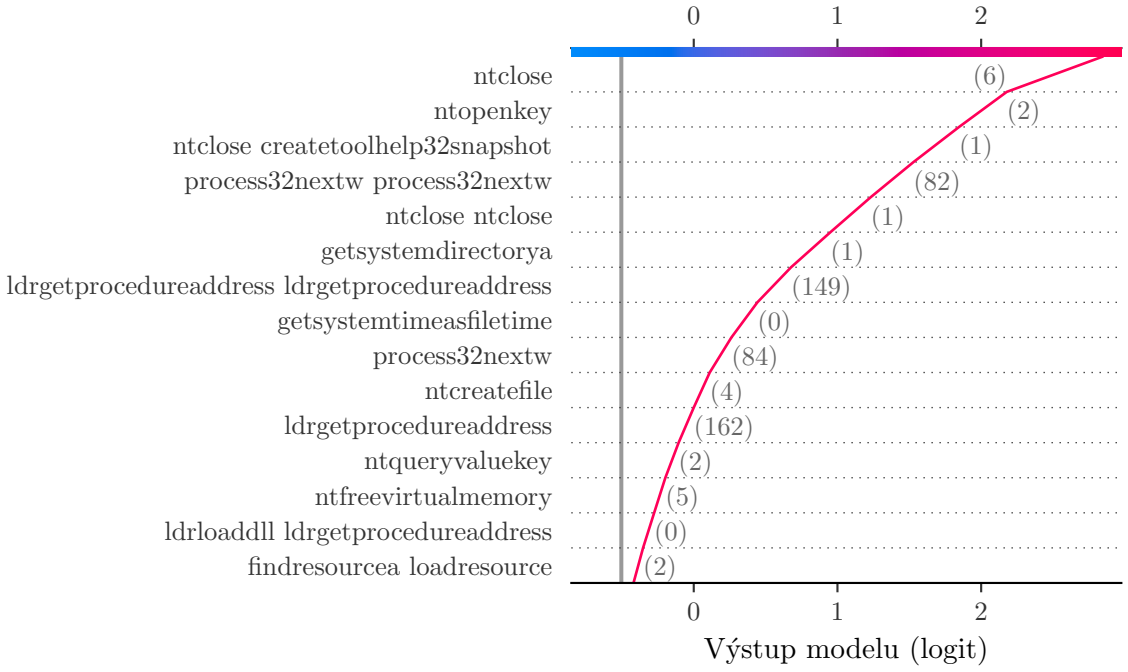
6.5 Vysvetlenia pre XGBoost

V tejto podkapitole sa zameriame na vysvetlenie rozhodnutí modelu XGBoost pri dvoch náhodne vybraných vzorkách, ktoré model správne klasifikoval. Jedna je z triedy spajvér (vzorka 1166) a druhá z triedy trojan (vzorka 843).

6.5.1 Vzorka 1166

V tejto časti analyzujeme vzorku 1166 patriacu do triedy spajvér. Podľa rozhodovacieho grafu SHAP (obrázok 10) má pre model najväčší prínos na predikcii prítomnosť viacerých volaní `NtClose` (6). Napriek tomu, že tieto volania poukazujú na predchádzajúcu manipuláciu so súbormi alebo registrami, samotné nie sú charakteristickým znakom spajvéru. Ich veľmi vysoký význam pre model preto môže naznačovať, že došlo k čiastočnému preučeniu na trénovacích dátach a model sa naučil rozpoznávať nesprávny signál pre triedu spajvér.

Volania `NtClose` pravdepodobne súvisia s uzatváraním deskriptorov súborov vytvorených cez `NtCreateFile` (4) a registrov otvorených pomocou `NtOpenKey` (2), keďže počty výskytov týchto volaní sa zhodujú. Spajvér môže `NtCreateFile` použiť napr. na dočasné uloženie získaných dát alebo snímok obrazovky, ktoré neskôr odošle útočníkovi. Čítanie z registrov môže zas slúžiť na detekciu sandbox prostredí či antivírusových riešení.



Obr. 10: Rozhodovací graf SHAP pre vzorku 1166 správne klasifikovanú modelom XGBoost ako spajvér

Z hľadiska indikácie typického správania spajvéru sú však najzaujímavejšie volania API funkcií `CreateToolhelp32Snapshot` (1) a `Process32NextW` (84). Funkcia `CreateToolhelp32Snapshot` sa používa na získanie prehľadu o aktuálnych systémových prostriedkoch, ako sú bežiace procesy, vlákna, načítané DLL alebo haldy. API volanie `Process32NextW` zas umožňuje iteráciu cez tieto položky a jeho prítomnosť môže naznačovať, že spajvér hľadá vhodné ciele na sledovanie, napríklad webové prehliadače alebo komunikačné aplikácie. Podozrenie vzbudzuje taktiež vysoký počet volaní pre dynamické získanie adries `LdrGetProcedureAddress` (162), ktorý môže byť použitý na prístup k súborom, injektovanie do procesov alebo na sieťovú komunikáciu. Celkový vzorec správania pôsobí ako silný indikátor prítomnosti spajvéru, ktorý model dokázal správne identifikovať.

Pri hľadaní vhodného pravidla, Anchors metóda narazila na maximálny počet predikátov a musela byť predčasne ukončená. Výsledné neúplné pravidlo (4) dosahuje

presnosť iba 0,164, čo ho robí nevhodným na vysvetlenie správania modelu pre danú triedu.

IF `NtAllocateVirtualMemory` > 0 **AND** `LdrGetProcedureAddress` > 0
AND `NtOpenKey` > 0 **AND** `IsProcessorFeaturePresent` $= 0$
AND `VirtualFree` $= 0$ **AND** `DefWindowProcA` $= 0$
AND `InternetOpenA` $= 0$ **AND** `GetProcAddress VirtualProtect` $= 0$
AND `QueryPerformanceCounter` $= 0$ **AND** `GetVersionExA` $= 0$
AND `GetLocaleInfoA` $= 0$ **AND** `GetDC` $= 0$ (4)
AND `ShowWindow` $= 0$ **AND** `memcpy memset` $= 0$
AND `PostQuitMessage` $= 0$
THEN PREDICT Dropper

Presnosť: 0,164 Pokrytie: 0,2359

Nájdený kontrafaktuál si pre zmenu klasifikácie z triedy spajvér vyžadoval väčší počet úprav príznakov. Takmer všetky upravené príznaky alebo ich n-gramy sa už objavili aj v rozhodovacom grafe modelu, hoci s odlišnou úrovňou dôležitosti. Medzi potrebnými zmenami je síce prítomný príznak `NtClose CreateToolhelp32Snapshot`, no chýba `Process32NextW` aj `LdrGetProcedureAddress`. To však nie je prekvapujúce, keďže cieľom kontrafaktuálnej metódy je nájsť čo najmenšie zmeny v príznakoch. Pravdepodobne by teda bolo potrebné výraznejšie zníženie ich výskytu, keďže ide o príznaky s vysokým informačným prínosom. Tým sa potvrdzuje, že model považuje tieto príznaky za kľúčové a je odolný voči čiastočným úpravám.

Tento efekt je viditeľný aj v kontrafaktuálnom pravdepodobnostnom rozdelení tried (tabuľka 22). Aj po úpravách vzorky zostáva spajvér jednou z najsilnejších tried (0,208), pričom kontrafaktuálna trieda červ získala len miernu preferenciu (0,245).

6.5.2 Vzorka 843

V tejto časti si rozoberieme vzorku 843 patriacu do triedy trojan. Na prvý pohľad je z rozhodovacieho grafu (obrázok 11) zrejmé, že model sa naučil rozpoznávať trojan prevážne na základe šumu v dátach.

Ako najdôležitejší príznak SHAP určil `atoi` (1). Ide o bežnú C funkciu na konverziu reťazcov na celé čísla. Táto funkcia je prítomná v mnohých legitímnych aplikáciách a sama o sebe nevyvoláva podozrenie.

Na druhom mieste sa umiestnilo API `GetProcAddress` (1), ktoré slúži na dynamické získanie adresy funkcie počas behu programu. Ide o wrapper nad nízkoúrovňovou `LdrGetProcedureAddress`, ktorý sa pri predchádzajúcich analýzach vzoriek už viackrát

Tabuľka 21: Kontrafaktuálne zmeny pre vzorku 1166 správne klasifikovanú modelom XGBoost ako spajvér

Príznak	**Hodnota**	
	Pôvodná	**Zmenená**
`FindResourceA LoadResource`	2	1
`GetSystemDirectoryA`	1	0
`LoadResource SizeofResource`	2	1
`NtAllocateVirtualMemory NtAllocateVirtualMemory`	2	1
`NtClose CreateToolhelp32Snapshot`	1	0
`NtClose NtClose`	1	0
`NtClose NtOpenKey`	1	0
`NtCreateFile NtCreateFile`	1	0
`NtFreeVirtualMemory`	5	4
`NtFreeVirtualMemory NtFreeVirtualMemory`	1	0

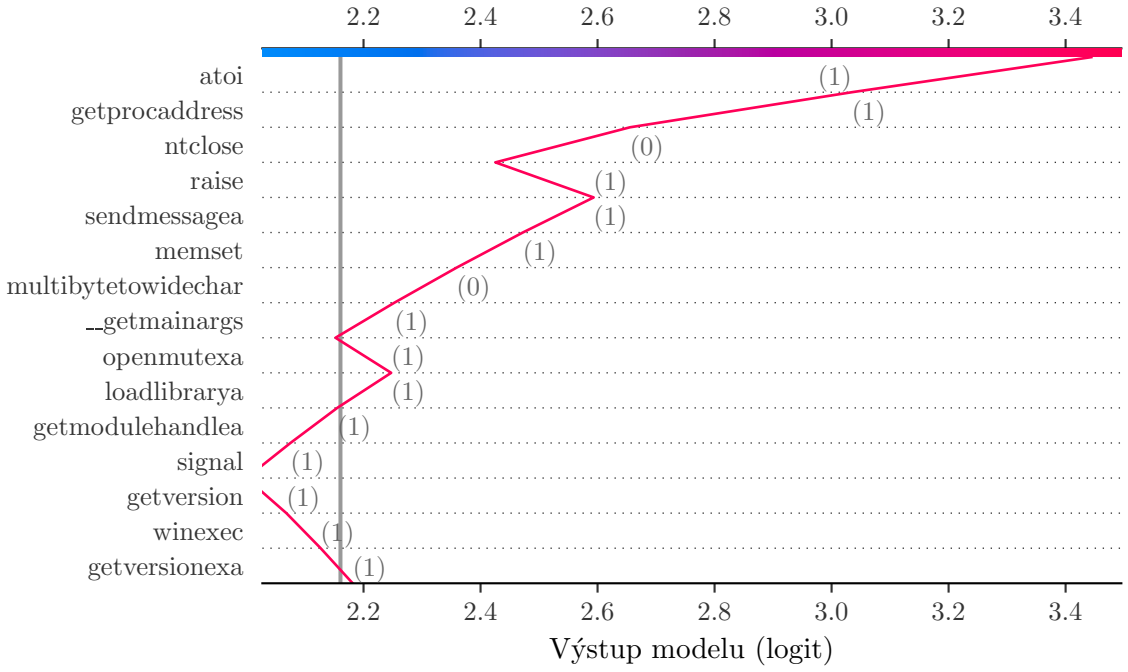
Tabuľka 22: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 1166 správne klasifikovanú modelom XGBoost ako spajvér

Vzorka	**Advér**	**Backdoor**	**Downloader**	**Dropper**	**Spajvér**	**Trojan**	**Vírus**	**Červ**
Pôvodná	0,002	0,029	0.020	0.061	**0,788**	0,059	0,004	0,038
Zmenená	0,008	0,117	0,073	0.172	0,208	0,158	0.020	**0,245**

objavil. Pokročilejší malvér preferuje priamy prístup cez `LdrGetProcedureAddress` primárne kvôli nižšej miere monitorovania, ktorej je vystavené. Volanie `GetProcAddress` je však bežné používané vo všetkých kategóriách malvéru a jeho prítomnosť preto nie je jednoznačným indikátorom triedy trojan.

Za pomerne dôležitý príznak označil SHAP aj *neprítomnosť* funkcie `NtClose`. Nevidíme však zrejmý dôvod, prečo by nepoužitie tejto funkcie malo súvisieť s triedou trojan.

Funkcia `SendMessageA` sa štandardne používa na interakciu s GUI. Ak však v danej vzorke žiadne grafické rozhranie nie je prítomné, môže to naznačovať, že cieľom volania je okno iného procesu, čo by mohlo predstavovať podozrivé správanie. V rozhodovacom grafe sa však nenachádzajú doplnkové volania ako `FindWindowA` alebo `SetWindowsHookEx`, ktoré by túto hypotézu podporili. Preto nie je možné určiť, či je prítomnosť `SendMessageA` skutočne významná.



Obr. 11: Rozhodovací graf SHAP pre vzorku 843 správne klasifikovanú modelom XGBoost ako trojan

Funkcia `LoadLibraryA`, ktorá sa v grafe taktiež objavuje, slúži na dynamické načítanie DLL. V kombinácii s `GetProcAddress` môže teda indikovať snahu o obídenie detekcie. Opäť sa však jedná o často používaný mechanizmus aj v iných typoch malvéru, a preto ho nemožno považovať za špecifický indikátor triedy trojan. Ostatné funkcie prítomné v grafe sú buď legitímne, alebo model od triedy trojan odkláňajú.

Metóda Anchors identifikovala rovnaké pravidlo (5) pre obe úrovne požadovanej presnosti, pričom dosiahla presnosť až 0,9927. Z pravidla vyplýva, že ak sa vo vzorke vyskytujú príznaky `GetProcAddress` a `signal`, tak s veľkou pravdepodobnosťou ho model zaradí do triedy trojan. Z doménového hľadiska však takéto pravidlo neposkytuje hodnotnú informáciu. Ako už bolo spomenuté, `GetProcAddress` sa vyskytuje vo viacerých typoch malvéru a `signal` je bežnou súčasťou C runtime prostredia.

IF `GetProcAddress` > 0 **AND** `signal` > 0
THEN PREDICT Trojan (5)

Presnosť: 0,9927 Pokrytie: 0,0637

Nájdený kontrafaktuál potvrdzuje závery predchádzajúcich metód. Na zmenu predikcie z trojan na backdoor bolo potrebné zmeniť hodnoty príznakov `atoi`, `GetProcAddress` a `RegCloseKey` z 1 na 0 (tabuľka 23). Oproti výsledkom z predchádzajúcich dvoch vysvetlení pribudol príznak `RegCloseKey`. Indikuje manipuláciu s registrami, čo môže

predstavovať podozrivé správanie, avšak opäť sa nejedná o unikátny vzorec pre triedu malvér.

Na základe celkovej analýzy je zrejmé, že model sa nenaučil rozpoznávať vzorce správania, ktoré by jednoznačne indikovali triedu trojan a zároveň zodpovedali doménovým poznatkom. Zdá sa, že jeho vysoká predikčná úspešnosť je spôsobená najmä prítomnosťou oveľa väčšieho počtu vzoriek než pre iné triedy. To mu umožnilo naučiť sa rozpoznávať trojan podľa šumu prítomného v dátach.

Tabuľka 23: Kontrafaktuálne zmeny pre vzorku 843 správne klasifikovanú modelom XGBoost ako trojan

	Hodnota	
Príznak	**Pôvodná**	**Zmenená**
`atoi`	1	0
`GetProcAddress`	1	0
`RegCloseKey`	1	0

Tabuľka 24: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 843 správne klasifikovanú modelom XGBoost ako trojan

Vzorka	**Advér**	**Backdoor**	**Downloader**	**Dropper**	**Spajvér**	**Trojan**	**Vírus**	**Červ**
Pôvodná	0,001	0,089	0.001	0.000	0.001	**0,908**	0,000	0,001
Zmenená	0,001	**0,573**	0,004	0.001	0,001	0,416	0.001	0.002

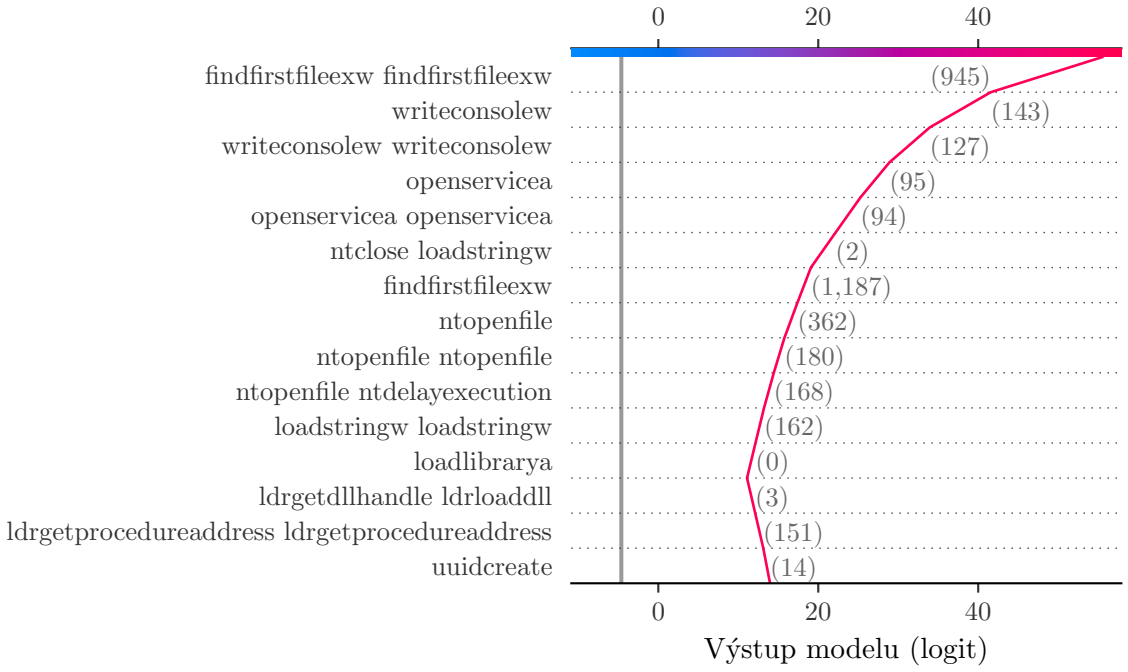
6.6 Vysvetlenia pre viacvrstvový perceptrón

V tejto podkapitole vykonáme analýzu na základe vysvetlení z modelu viacvrstvového perceptrónu na dvoch náhodne vybraných vzorkách, ktoré boli správne klasifikované. Prvá patrí do kategórie vírus (vzorka 1089) a druhá do kategórie dropper (272).

6.6.1 Vzorka 1089

V tejto časti si analyzujeme vzorku 1089 z triedy vírus. Z rozhodovacieho grafu SHAP (obrázok 12) vyplýva, že za najdôležitejšie príznaky považuje `FindFirstFileExW`, `WriteConsoleW`, `OpenServiceA` a ich bigramy.

API `FindFirstFileExW`, bežne používané na vyhľadávanie súborov, vykazuje v tomto prípade mimoriadne vysoký výskyt (1187). Môže indikovať rekurzívne skenovanie



Obr. 12: Rozhodovací graf SHAP pre vzorku 1166 správne klasifikovanú modelom MLP ako vírus

priečinkov, napr. za účelom hľadania vhodných cieľov na infikovanie, identifikáciu spustiteľných súborov alebo detekciu antivírusového softvéru.

Podozrivým je aj opakované používanie nízkoúrovňového volania `NtOpenFile` (362), ktoré slúži na otváranie súborových deskriptorov. V kombinácii s `FindFirstFileExW` to naznačuje systematické otváranie nájdených súborov. Použitie nízkoúrovňového volania, namiesto bežného `CreateFile`, môže indikovať pokus o obídenie monitorovania API vyššej úrovne. Model sa však správne naučil tento vzorec detegovať.

Aj keď samotné otvorenie deskriptoru ešte neznamená čítanie či zápis, model zachytil vysoký výskyt bigramov `LdrGetProcedureAddress` (151). Pomocou nich môže vírus dynamicky vykonať práve túto funkcionalitu a vyhnúť sa pritom statickej detekcii. Zaujímavé však je, že podľa SHAP tento príznak znižuje pravdepodobnosť zaradenia do triedy vírus.

Obzvlášť podozrivým je aj bigram `NtOpenFile NtDelayExecution` (168), ktorý vykonáva pozastavenie činnosti hneď po otvorení súborového deskriptoru. To naznačuje ďalšiu taktiku na vyhnutie sa detekcii, tentokrát sandboxovej analýzy.

SHAP taktiež identifikoval ako dôležité veľké množstvo volaní `WriteConsoleW` (143), ktoré slúži na výstup do konzoly. Táto funkcionalita však sama o sebe nie je nebezpečná a nezdá sa byť ani ničím typická pre vírusy. Model sa teda pravdepodobne naučil nesprávne považovať tento príznak za dôležitý indikátor prítomnosti vírusu.

Metóda Anchors našla jedno pravidlo (6) s vysokou presnosťou 0,9692, avšak s nulovým pokrytím. To znamená, že pravidlo sa vzťahuje na tak malý počet vzoriek (možno len na túto konkrétnu), že pri náhodnom výbere vzoriek z podkladového datasetu pre odhad pokrytia sa zhoda neobjavila ani raz. To potvrdzuje náš predpoklad, že vysoká dôležitosť `WriteConsoleW` je chybným pozitívnym signálom pre triedu vírus. Anchors sa na ňom však zasekne, keďže má silnú indikáciu pre cieľovú triedu. Umožnenie hľadania viacerých kandidátov by tento problém mohol odstrániť, no ako sme už uviedli, tak pre výpočtové nároky to nebolo uskutočniteľné.

IF `WriteConsoleW WriteConsoleW` > 0
AND `NtOpenFile NtDelayExecution` > 0
THEN PREDICT Vírus

Presnosť: 0,9692 Pokrytie: 0,000 (6)

Kontrafaktuálna analýza takisto potvrdzuje význam `WriteConsoleW` pre finálnu predikciu (tabuľka 25). Zmena triedy z vírus na červ bola možná len po úplnom odstránení tohto príznaku.

Tabuľka 25: Kontrafaktuálne zmeny pre vzorku 1089 správne klasifikovanú modelom MLP ako vírus

	Hodnota	
Príznak	**Pôvodná**	**Zmenená**
`CreateThread`	6	0
`GetFileType NtCreateFile`	0	441
`WriteConsoleW`	143	0
`WSAStartup`	0	84

Okrem toho sa ukázala potreba vynulovať aj príznak `CreateThread`. Ten sa v žiadnom z predchádzajúcich vysvetlení neobjavil, napriek tomu, že ide o obzvlášť podozrivé volanie. Môže byť použité na spustenie shell kódu vytvorením nového vlákna, ktoré vykonáva inštrukcie z pamäťovej oblasti, do ktorej malvér predtým injektoval kód. Ďalšie úpravy, ako zvýšenie výskytu `GetFileType NtCreateFile` a `WSAStartup`, už viac odrážajú znaky typické pre triedu červ než pre vírus.

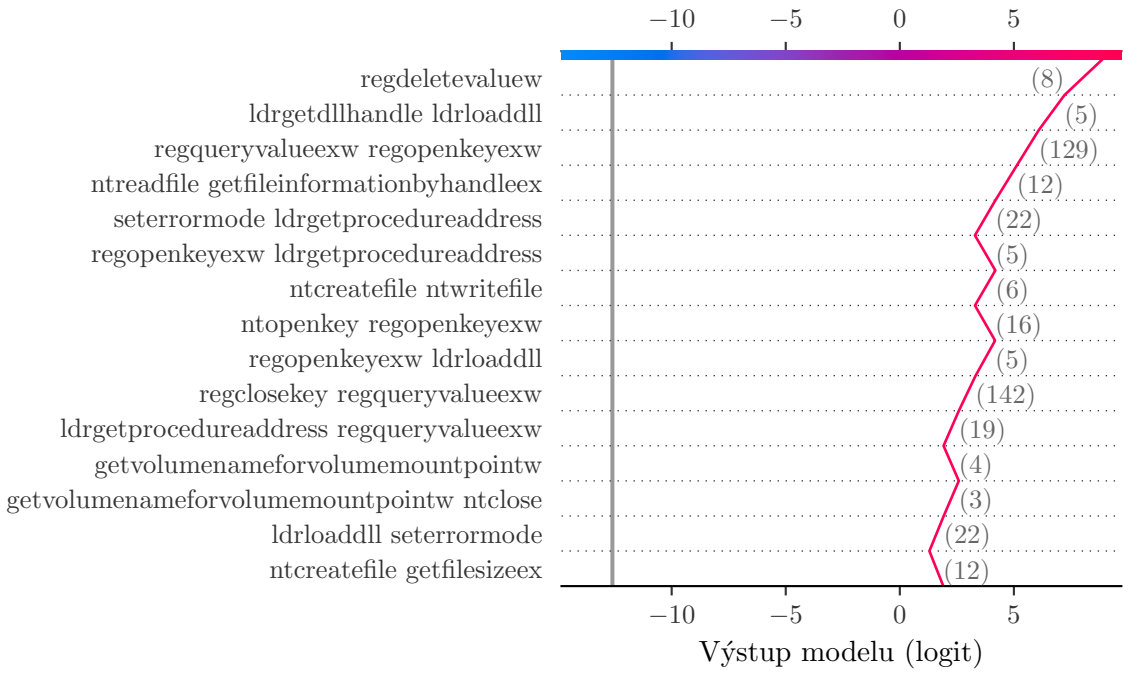
Aj po týchto zmenách zostáva predikcia len tesne naklonená v prospech triedy červ (tabuľka 26). To naznačuje, že napriek silnému signálu z `WriteConsoleW`, model od nej nie je závislý a vníma aj širší kontext správania typického pre vírusy.

Tabuľka 26: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 1089 správne klasifikovanú modelom MLP ako vírus

Vzorka	Advér	Backdoor	Downloader	Dropper	Spajvér	Trojan	Vírus	Červ
Pôvodná	0,000	0,000	0,000	0,000	0,000	0,000	**1,000**	0,000
Zmenená	0,000	0,000	0,000	0,000	0,000	0,000	0,438	**0,562**

6.6.2 Vzorka 272

V tejto časti si bližšie rozoberieme vzorku 272 klasifikovanú ako dropper. Rozhodovací graf (obrázok 13) poukazuje na vysoký výskyt podozrivých volaní súvisiacich s registrami, ako napríklad `RegDeleteValueW` na odstránenie hodnoty registra, `RegOpenKeyExW` na otvorenie kľúča registra, `RegQueryValueExW` na jeho čítanie a `RegCloseKey` na jeho uzavretie.



Obr. 13: Rozhodovací graf SHAP pre vzorku 272 správne klasifikovanú modelom MLP ako dropper

Medzi volaniami však chýbajú funkcie určené na úpravu alebo vytváranie registrov, ako napríklad `RegSetValueExW` či `RegCreateKeyExW`. Pri bližšom pohľade na prítomné volania si je však možné všimnúť viacero bigramov obsahujúcich, pre nás už dobre známe, volania `LdrGetProcedureAddress`. Tie naznačujú snahu vykonávať modifikáciu registrov dynamicky a tým sa vyhnúť detekcii. Priame používanie API ako `RegSetValueExW` či `RegCreateKeyExW` je totiž často intenzívne monitorované bezpečnostným softvérom.

Takéto správanie je teda mimoriadne podozrivé a môže signalizovať viacero zámerov. Jedným z nich môže byť nastavenie droppera na automatické spustenie pri štarte systému, čo by umožnilo opätovnú inštaláciu škodlivého obsahu v prípade jeho odstránenia. Ďalšou možnosťou je vypnutie ochrán systému, ako napríklad Windows Defender, alebo detekcia sandbox prostredia, v ktorom dropper následne nevykoná škodlivú činnosť, aby sa vyhol detekcii.

Kľúčovým znakom pre klasifikáciu do triedy dropper je pre nás predovšetkým viacero nízkoúrovňových súborových operácií prostredníctvom volaní `NtCreateFile NtWriteFile` (6) a `NtCreateFile GetFileSizeEx` (12). Tieto sú pre dropper typické, keďže jeho primárnou úlohou je zavedenie iného malvéru do systému. Zaujímavé je však, že podľa SHAP má prítomnosť volania `NtCreateFile GetFileSizeEx` skôr negatívny efekt pre klasifikáciu ako dropper. Aj napriek tomu model preukázal schopnosť správne identifikovať vzorec správania typický pre dropper, hoci sa vzorka snaží vyhnúť detekcii.

Metóda Anchors identifikovala pravidlá pre obe verzie presnosti. V druhej verzii však došlo k predčasnému ukončeniu kvôli dosiahnutiu limitu predikátov ešte pred dosiahnutím požadovanej presnosti 0,9.

Prvá verzia pravidla (7) dosiahla presnosť 0,8057 a pokrytie 0,012. Potvrdila dôležitosť volaní na vytváranie a zápis do súborov prostredníctvom `NtCreateFile NtWriteFile`. Objavila takisto potrebu výskytu volania `NtMapViewOfSection`, ktoré je droppermi často využívané na mapovanie škodlivého kódu priamo do pamäte s cieľom vyhnúť sa detekcii pri zápise na disk. Nečakaným je však prítomnosť predikátu vyžadujúceho nulový výskyt volania `RegQueryValueExW`, aj napriek tomu, že model jeho prítomnosť v bigramoch považuje za silný pozitívny signál. To môže indikovať, že model si nedokázal vytvoriť dostatočný korelačný vzťah medzi samotným volaním a jeho bigramami.

IF `NtClose NtMapViewOfSection` > 0 **AND** `NtCreateFile NtWriteFile` > 0
AND `RegQueryValueExW` $= 0$ **AND** `GetSystemMetrics GetCursorPos` $= 0$
AND `OpenServiceA` $= 0$ **AND** `GetSystemInfo NtAllocateVirtualMemory` $= 0$
AND `VirtualAlloc` $= 0$ **AND** `NtDelayExecution FindFirstFileExW` $= 0$
AND `FindWindowA NtOpenProcess` $= 0$ **AND** `GetCursorPos NtClose` $= 0$
AND `LdrGetProcedureAddress FindResourceExW` $= 0$
AND `DeleteFileW GetSystemDirectoryA` $= 0$
THEN PREDICT Dropper

Presnosť: 0,8057 Pokrytie: 0,012

(7)

Druhá verzia pravidla (8) dosiahla vyššiu presnosť 0,8423, ale veľmi nízke pokrytie 0,0006. Okrem predchádzajúcich predikátov pribudli tri nové, z ktorých najzaujímavejším je požiadavka na nulový výskyt bigramu `RegCreateKeyExA RegCloseKey`. Na základe výstupov SHAP by sme očakávali, že tieto volania klasifikáciu ako dropper skôr podporia. Dôvodom je pravdepodobne nedostatok takýchto príkladov v trénovacích dátach, keďže množstvo malvérových vzoriek zápis do registrov realizuje dynamicky, aby sa tak vyhli detekcii. Paradoxne tak model rozpoznáva skôr absenciu týchto volaní ako ich výskyt.

IF `NtClose NtMapViewOfSection` > 0 **AND** `NtCreateFile NtWriteFile` > 0
AND `RegQueryValueExW` $= 0$ **AND** `GetSystemMetrics GetCursorPos` $= 0$
AND `OpenServiceA` $= 0$ **AND** `GetSystemInfo NtAllocateVirtualMemory` $= 0$
AND `VirtualAlloc` $= 0$ **AND** `NtDelayExecution FindFirstFileExW` $= 0$
AND `FindWindowA NtOpenProcess` $= 0$ **AND** `GetCursorPos NtClose` $= 0$
AND `LdrGetProcedureAddress FindResourceExW` $= 0$
AND `DeleteFileW GetSystemDirectoryA` $= 0$
AND `NtClose GetFileAttributesW` $= 0$ **AND** `UuidCreate` $= 0$
AND `RegCreateKeyExA RegCloseKey` $= 0$
THEN PREDICT Dropper

Presnosť: 0,8423 Pokrytie: 0,0006

(8)

Zaujímavým zistením z kontrafaktuálnej analýzy (tabuľka 27) je, že pre danú vzorku je najbližšou, z hľadiska minimalizácie zmien, trieda backdoor. Toto je pomerne prekvapujúce, keďže prirodzene by sme očakávali, že najbližšia trieda bude downloader kvôli ich značnej podobnosti vo funkcionalite. Identifikovaná kontrafaktuálna zmena bolo zvýšenie výskytu `WSAStartup` z 0 na 87. Takáto zmena však nie je realistická, keďže zvyčajne stačí `Winsock` inicializovať iba raz.

Tabuľka 27: Kontrafaktuálne zmeny pre vzorku 272 správne klasifikovanú modelom MLP ako dropper

	Hodnota	
Príznak	**Pôvodná**	**Zmenená**
`WSAStartup`	0	87

Dôležitým zistením ale je, že model zrejme nezachytil *neprítomnosť* sieťovej komunikácie ako charakteristickú črtu droppera. To sa odráža v tom, že na zmenu klasifikácie bolo potrebné výrazné zvýšenie hodnoty tohto volania. Aj napriek tejto zmene však

zostala pravdepodobnosť pre triedu dropper pomerne vysoká (tabuľka 28). To naznačuje, že signál z `WSAStartup` model nepovažuje za veľmi dôležitý.

Tabuľka 28: Pravdepodobnostné rozloženie tried pred a po aplikácií kontrafaktuálnych zmien pre vzorku 272 správne klasifikovanú modelom MLP ako dropper

Vzorka	**Advér**	**Backdoor**	**Downloader**	**Dropper**	**Spajvér**	**Trojan**	**Vírus**	**Červ**
Pôvodná	0,000	0,000	0,000	**0,994**	0,006	0,000	0,000	0,000
Zmenená	0,000	**0,403**	0,000	0,397	0,198	0,000	0,000	0,002

6.7 Porovnanie a vyhodnotenie vysvetľovacích metód

Aplikácia vysvetľovacích metód SHAP, Anchors a kontrafaktuálnych vysvetlení nám umožnila lepšie porozumieť správaniu jednotlivých klasifikačných modelov pri rôznych vstupoch, ako aj posúdiť vhodnosť týchto metód pre oblasť detekcie kategórií malvéru.

Metóda SHAP, konkrétne jej rozhodovací graf, sa ukázal z hľadiska informačnej hodnoty ako najprínosnejší. Poskytuje totiž detailný, no zároveň prehľadný pohľad na to, ktoré príznaky model považuje za najdôležitejšie pri rozhodovaní. Obzvlášť sme ocenili jeho schopnosť vyjadriť presný príspevok jednotlivých príznakov k výslednej predikcii a usporiadať ich podľa dôležitosti.

Najmä vďaka SHAP sme boli schopní jednoznačne identifikovať detekčné vzorce modelov a tiež ich nedostatky. Ako príklad možno uviesť analýzu vzorky 843 (časť 6.5.2), kde sa ukázalo, že modely pre kategóriu trojan nemajú dostatočne reprezentatívne dáta a svoje rozhodnutia zakladajú prevážne na šume v dátach. Za výhodu tiež možno považovať relatívne nízku výpočtovú náročnosť, najmä pri použití optimalizovaných algoritmov pre konkrétne typy modelov.

Kde však táto metóda zlyháva, je odhad správania modelu pri modifikácii vstupných príznakov. Výstupné vysvetlenia sú totiž platné iba pre konkrétne hodnoty príznakov. V opačnom prípade nie je možné iba na základe rozhodovacieho grafu spoľahlivo určiť, akým smerom a ako moc sa predikcia pohne. Tento nedostatok sme spozorovali napr. na vzorke 170 (časť 6.4.2).

Na vyriešenie tohto problému sa osvedčilo použitie pravidiel z metódy Anchors. Splnením výstupnej množiny predikátov je totiž garantované (s určitou presnosťou) zachovanie rovnakej predikcie, pričom poskytuje aj odhad pokrytia. Táto metóda sa ukázala ako mimoriadne užitočná pri overovaní, či model zakladá svoje rozhodnutia na skutočne relevantných príznakoch. Pri dobre rozpoznateľných kategóriách, ako je napr. advér vo vzorke 187 (časť 6.4.1), boli anchor pravidlá stabilné, jednoznačné a reflektovali známe doménové poznatky.

Zrejme najväčšou nevýhodou metódy Anchors je však jej vysoká výpočtová náročnosť, obzvlášť pri úlohách s veľkým počtom príznakov, čo je typické práve pre oblasť detekcie kategórií malvéru. Boli sme preto nútení nastaviť obmedzenie na maximálny počet predikátov. Metóda Anchors, pri viacerých vzorkách na tento limit narazila a nebola tak schopná poskytnúť užitočné pravidlá. Navyše, metóda poskytuje najlepšie výsledné pravidlá len pri paralelnom hľadaní viacerých kandidátov, čo však opäť náramne zvyšuje výpočtovú záťaž.

Kontrafaktuálne vysvetlenia predstavujú odlišný prístup. Neodpovedajú totiž na otázku *prečo* model spravil dané rozhodnutie, ale *čo* by sa muselo zmeniť, aby bol výsledok iný. Tento typ vysvetlenia sa ukázal byť ako mimoriadne prínosný pri skúmaní hraníc modelov. Umožnil nám totiž určiť citlivosť modelu na jednotlivé príznaky a odhaliť, či aj malé zmeny dokážu viesť k zásadnej odchýlke vo výstupe.

Samotná metóda kontrafaktuálnych vysvetlení však neposkytuje dostatočne komplexný pohľad na správanie modelu. Zameriava sa totiž len na upravené príznaky a podobne ako SHAP, je závislá na konkrétnych vstupných príznakoch. Preto sa pre náš účel osvedčila skôr ako doplnková metóda.

Na záver možno konštatovať, že žiadna z vysvetľovacích metód sa neukázala ako samostatne postačujúca. Každá z nich má svoje výhody, ale aj obmedzenia. Na získanie komplexného pohľadu na správanie modelov je preto vhodné použiť všetky tri metódy súčasne, keďže iba tak je možné dosiahnuť skutočne kvalitné a dôveryhodné vysvetlenia.

Záver

V tejto práci sme sa zaoberali vysvetliteľnou klasifikáciou kategórií malvéru na základe dát z dynamickej analýzy. Implementovali sme tri modely strojového učenia, založené na algoritmoch náhodný les, XGBoost a viacvrstvový perceptrón. Na tieto modely sme následne aplikovali vysvetľovacie metódy SHAP, Anchors a kontrafaktuálne vysvetlenia, pričom sme vyhodnotili ich vhodnosť pre tento typ úlohy.

Dôležitou súčasťou práce bol aj výber vhodného datasetu pre trénovanie modelov. Po analýze viacerých existujúcich datasetov sme však zistili, že žiaden z nich nespĺňa všetky naše požiadavky (podkapitola 4.1). Preto sme pristúpili ku kombinácii a predspracovaniu dvoch z nich, čím vznikol nový dataset, ktorý už zodpovedal našim potrebám.

V priebehu práce sme zdokumentovali celý proces implementácie klasifikačných modelov, vrátane ladenia hyperparametrov vektorizácie aj samotných modelov. Výsledky 10-násobnej krížovej validácie ukázali, že model náhodného lesa dosiahol F1-skóre 0,717, model XGBoost 0,733 a viacvrstvový perceptrón 0,687 (podkapitola 5.6). Najlepšie výsledky teda dosiahol model XGBoost, hoci náhodný les mu bol výkonom pomerne blízko. Viacvrstvový perceptrón zaostal najmä kvôli nutnosti obmedzenia počtu príznakov z výpočtových dôvodov. Podarilo sa nám prekonať výsledky referenčných modelov jedného z použitých datasetov, pričom model XGBoost sa zlepšil o 0,015 bodu a náhodný les dokonca až o 0,115 bodu. Túto časť teda hodnotíme ako úspešnú.

V záverečnej časti práce sme aplikovali vysvetľovacie metódy na náhodne vybrané vzorky pre každý model, na základe čoho sme posúdili ich praktickú použiteľnosť (podkapitola 6.7). Najinformatívnejšou metódou sa ukázala byť SHAP, ktorá nám poskytla detailný prehľad o vplyve jednotlivých príznakov na rozhodovanie modelu. Umožnila nám tiež identifikovať detekčné vzorce modelov pre konkrétne kategórie malvéru. Nevýhodou tejto metódy je však jej nižšia stabilita pri zmene vstupných príznakov. V tomto ohľade sa nám ako vhodnejšia ukázala metóda Anchors. Tá generuje pravidlá zložené z množiny predikátov, pričom ich splnenie zaručuje zachovanie predikcie a ponúka aj odhad rozsahu ich platnosti. Jej hlavnou nevýhodou je vysoká výpočtová náročnosť, čo je problém najmä pri modeloch s veľkým počtom príznakov. Tie sú však typické pre detekciu kategórií malvéru na základe dynamickej analýzy. Poslednou použitou metódou boli kontrafaktuálne vysvetlenia, ktoré informujú, aké zmeny v príznakoch by viedli k inej predikcii. Táto metóda sa nám ukázala ako užitočná najmä na posúdenie citlivosti modelov na jednotlivé príznaky, no na komplexné vysvetlenie správania modelov nestačí. Pre získanie uceleného pohľadu na klasifikáciu kategórií malvéru sa preto ukázalo ako potrebné skombinovať všetky tri vysvetľovacie metódy.

Aplikácia vysvetľovacích metód nám zároveň umožnila odhaliť nedostatky v správaní modelov. Ako problematická sa ukázala byť kategória trojan. Hoci modely dosahovali pri jej detekcii vysokú úspešnosť, vysvetľovacie metódy odhalili, že pri rozpoznávaní tejto kategórie sa opierajú o, z pohľadu domény, nerelevantné informácie. Ako hlavný dôvod sa ukázala byť prítomnosť funkcií z C/C++ runtime knižníc a iných systémových knižníc v dátach. Napriek nášmu pôvodnému predpokladu, neprispeli k zlepšeniu klasifikácie, ale iba zvýšili dátový šum. Bolo by preto vhodné dataset manuálne prečistiť a ponechať len relevantné funkcie z Windows API a následne modely opätovne natrénovať. Vzhľadom na rozsiahly počet unikátnych príznakov (až 10 513) a nekompletný zoznam dostupných Windows API funkcií ide o zaujímavú praktickú výzvu, ktorá by mohla významne prispieť k zlepšeniu výsledkov klasifikačných modelov.

Literatúra

1. OORSCHOT, P. C. van. *Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin*. 2. vyd. Springer Cham, 2021. ISBN 978-3-030-83411-1. Dostupné z DOI: https://doi.org/10.1007/978-3-030-83411-1.
2. INVESTIGATION, F. B. of. *2020 Internet Crime Report* [online]. 2021. [cit. 2025-02-10]. Dostupné z: https://www.ic3.gov/AnnualReport/Reports/2020_IC3Report.pdf.
3. *What Is a Drive by Download* [online]. Kaspersky Lab, 2025 [cit. 2025-02-10]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/drive-by-download.
4. AYCOCK, J. *Spyware and Adware*. 1. vyd. Springer New York, NY, 2010. ISBN 978-0-387-77741-2. Dostupné z DOI: https://doi.org/10.1007/978-0-387-77741-2.
5. *Court Rules That Zango Can't Sue Kaspersky Over Blocked 'Adware'* [online]. Informa TechTarget, 2009 [cit. 2025-02-10]. Dostupné z: https://www.darkreading.com/cyber-risk/court-rules-that-zango-can-t-sue-kaspersky-over-blocked-adware-.
6. *What is incident response?* [online]. IBM, 2025 [cit. 2025-02-20]. Dostupné z: https://www.ibm.com/think/topics/dfir.
7. SIKORSKI, M.; HONIG, A. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012. ISBN 9781593272906. Dostupné tiež z: https://books.google.sk/books?id=AD-QEAAAQBAJ.
8. *Process Injection: Process Hollowing* [online]. MITRE ATT&CK, 2020-01-14 [cit. 2025-02-25]. Dostupné z: https://attack.mitre.org/techniques/T1055/012/.
9. *Process Injection: Asynchronous Procedure Call* [online]. MITRE ATT&CK, 2020-01-14 [cit. 2025-02-25]. Dostupné z: https://attack.mitre.org/techniques/T1055/004/.
10. LOUPPE, G. *Understanding Random Forests: From Theory to Practice*. 2015. Dostupné z arXiv: 1407.7502 [stat.ML].
11. CHA ZHANG, Y. M. *Ensemble Machine Learning: Methods and Applications*. 1. vyd. Springer New York, NY, 2012. ISBN 978-1-4419-9326-7. Dostupné z DOI: https://doi.org/10.1007/978-1-4419-9326-7.

12. PROBST, P.; WRIGHT, M. N.; BOULESTEIX, A.-L. Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*. 2019, **9**(3). ISSN 1942-4795. Dostupné z DOI: 10.1002/widm.1301.

13. *Permutation feature importance* [online]. scikit-learn, 2025 [cit. 2025-03-05]. Dostupné z: https://scikit-learn.org/stable/modules/permutation_importance.html.

14. MA, E. J. Random Forests: A Good Default Model? *Eric J. Ma's Blog* [https://ericmjl.github.io]. 2017. Dostupné tiež z: https://ericmjl.github.io/blog/2017/10/27/random-forests-a-good-default-model.

15. CHEN, T.; GUESTRIN, C. XGBoost: A Scalable Tree Boosting System. *CoRR*. 2016, **abs/1603.02754**. Dostupné z arXiv: 1603.02754.

16. *What is XGBoost?* [online]. IBM, 2024-05-09 [cit. 2025-02-27]. Dostupné z: https://www.ibm.com/think/topics/xgboost.

17. *Machine Learning Challenge Winning Solutions* [online]. xgboost [cit. 2025-02-27]. Dostupné z: https://github.com/dmlc/xgboost/blob/master/demo/README.md#machine-learning-challenge-winning-solutions.

18. *XGBoost Best Feature Importance Score* [online]. XGBoosting, 2024 [cit. 2025-03-01]. Dostupné z: https://xgboosting.com/xgboost-best-feature-importance-score.

19. SABRY, F. *Multilayer Perceptron: Fundamentals and Applications for Decoding Neural Networks*. One Billion Knowledgeable, 2023. Artificial Intelligence. Dostupné tiež z: https://books.google.sk/books?id=QFXHEAAAQBAJ.

20. BROWNLEE, J.; TAM, A.; HEIKKINEN, D.; LAM, A.; SETHI, D. *Deep Learning with PyTorch: Learn Basic Deep Learning with Minimal Code in PyTorch 2.0*. Machine Learning Mastery, 2023. Dostupné tiež z: https://books.google.sk/books?id=sSC1EAAAQBAJ.

21. SHOOK, J.; SMITH, R.; ANTONIO, A. Transparency and Fairness in Machine Learning Applications. *Texas A&M Journal of Property Law*. 2018, **4**(5), 443–463. Dostupné z DOI: 10.37419/JPL.V4.I5.2.

22. FLORA, M.; POTVIN, C.; MCGOVERN, A.; HANDLER, S. *Comparing Explanation Methods for Traditional Machine Learning Models Part 1: An Overview of Current Methods and Quantifying Their Disagreement*. 2022. Dostupné z arXiv: 2211.08943 [stat.ML].

23. MOLNAR, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 3. vyd. 2025. ISBN 978-3-911578-03-5. Dostupné tiež z: https://christophm.github.io/interpretable-ml-book.

24. DU, M.; LIU, N.; HU, X. *Techniques for Interpretable Machine Learning*. 2019. Dostupné z arXiv: 1808.00033 [cs.LG].

25. LUNDBERG, S. M.; LEE, S.-I. A Unified Approach to Interpreting Model Predictions. In: GUYON, I. et al. (ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, zv. 30. Dostupné tiež z: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

26. *Kernel SHAP* [online]. Seldon Technologies, 2019 [cit. 2025-03-05]. Dostupné z: https://docs.seldon.io/projects/alibi/en/latest/methods/KernelSHAP.html.

27. LUNDBERG, S. M. et al. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*. 2020, **2**(1), 2522–5839.

28. *Tree SHAP* [online]. Seldon Technologies, 2019 [cit. 2025-03-06]. Dostupné z: https://docs.seldon.io/projects/alibi/en/latest/methods/TreeSHAP.html.

29. SHRIKUMAR, A.; GREENSIDE, P.; KUNDAJE, A. *Learning Important Features Through Propagating Activation Differences*. 2019. Dostupné z arXiv: 1704.02685 [cs.CV].

30. LUNDBERG, S. *shap.DeepExplainer* [online]. 2018. [cit. 2025-03-10]. Dostupné z: https://shap.readthedocs.io/en/latest/generated/shap.DeepExplainer.html.

31. RIBEIRO, M.; SINGH, S.; GUESTRIN, C. Anchors: High-Precision Model-Agnostic Explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2018, **32**. Dostupné z DOI: 10.1609/aaai.v32i1.11491.

32. GUIDOTTI, R. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*. 2022, **38**, 1–55. Dostupné z DOI: 10.1007/s10618-022-00831-6.

33. WACHTER, S.; MITTELSTADT, B. D.; RUSSELL, C. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *CoRR*. 2017, **abs/1711.00399**. Dostupné z arXiv: 1711.00399.

34. LOOVEREN, A. V.; KLAISE, J. Interpretable Counterfactual Explanations Guided by Prototypes. *CoRR*. 2019, **abs/1907.02584**. Dostupné z arXiv: 1907.02584.

35. LAUGEL, T.; LESOT, M.-J.; MARSALA, C.; RENARD, X.; DETYNIECKI, M. *Inverse Classification for Comparison-based Interpretability in Machine Learning*. 2017. Dostupné z arXiv: 1712.08443 [stat.ML].

36. POYIADZI, R.; SOKOL, K.; SANTOS-RODRIGUEZ, R.; DE BIE, T.; FLACH, P. FACE: Feasible and Actionable Counterfactual Explanations. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. ACM, 2020, s. 344–350. AIES '20. Dostupné z DOI: 10.1145/3375627.3375850.

37. GUIDOTTI, R. et al. *Local Rule-Based Explanations of Black Box Decision Systems*. 2018. Dostupné z arXiv: 1805.10820 [cs.AI].

38. SYEDA, D. Z.; ASGHAR, M. N. Dynamic Malware Classification and API Categorisation of Windows Portable Executable Files Using Machine Learning. *Applied Sciences*. 2024, **14**(3). ISSN 2076-3417. Dostupné z DOI: 10.3390/app14031015.

39. SYEDA, D. Z.; ASGHAR, M. N. *Windows-PE-Malware-API-dataset*. 2024. Dostupné tiež z: https://github.com/drzehra14/Windows-PE-Malware-API-dataset.

40. DÜZGÜN, B. et al. *Benchmark Static API Call Datasets for Malware Family Classification*. 2022. Dostupné z arXiv: 2111.15205 [cs.CR].

41. DÜZGÜN, B. et al. *api_sequences_malware_datasets*. 2021. Dostupné tiež z: https://github.com/khas-ccip/api_sequences_malware_datasets.

42. CATAK, F. O.; AHMED, J.; SAHINBAS, K.; KHAND, Z. H. Data augmentation based malware detection using convolutional neural networks. *PeerJ Computer Science*. 2021, **7**, e346. ISSN 2376-5992. Dostupné z DOI: 10.7717/peerj-cs.346.

43. CATAK, F. O.; AHMED, J.; SAHINBAS, K.; KHAND, Z. H. *malware_api_class*. 2021. Dostupné tiež z: https://github.com/ocatak/malware_api_class.

44. *Name mangling (C++ only)* [online]. IBM, 2025 [cit. 2025-03-25]. Dostupné z: https://www.ibm.com/docs/en/i/7.6.0?topic=linkage-name-mangling-c-only.

45. *What Is Cross-Validation in Machine Learning?* [online]. Coursera, 2025 [cit. 2025-04-02]. Dostupné z: https://www.coursera.org/articles/what-is-cross-validation-in-machine-learning.

46. *Thresholds and the confusion matrix* [online]. Google, 2025 [cit. 2025-04-02]. Dostupné z: https://developers.google.com/machine-learning/crash-course/classification/thresholding.

47. *Classification: Accuracy, recall, precision, and related metrics* [online]. Google, 2025 [cit. 2025-04-02]. Dostupné z: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

48. JURAFSKY, D.; MARTIN, J. H. N-gram Language Models. In: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. 2025. Dostupné tiež z: https://web.stanford.edu/~jurafsky/slp3/3.pdf.

49. *BalancedRandomForestClassifier* [online]. imbalanced-learn, 2024 [cit. 2025-04-05]. Dostupné z: https://imbalanced-learn.org/stable/references/generated/imblearn.ensemble.BalancedRandomForestClassifier.html.

50. PROBST, P.; BOULESTEIX, A.-L. *To tune or not to tune the number of trees in random forest?* 2017. Dostupné z arXiv: 1705.05654 [stat.ML].

51. *XGBoost Parameters* [online]. xgboost, 2022 [cit. 2025-04-09]. Dostupné z: https://xgboost.readthedocs.io/en/stable/parameter.html.

52. *Most Important XGBoost Hyperparameters to Tune* [online]. XGBoosting, 2024 [cit. 2025-04-09]. Dostupné z: https://xgboosting.com/most-important-xgboost-hyperparameters-to-tune.

Dodatok A: Súborová štruktúra projektu

/xmc

· koreňový adresár

/xmc/src/xmc/explainers/explanations

· adresár pre vygenerované vysvetlenia pre metódy Anchors a kontrafaktuály

/xmc/src/xmc/plots

· adresár pre vygenerované grafy, vrátane SHAP vysvetlení

Dodatok B: Inštalačná a používateľská príručka

Zdrojový kód aplikácie je dostupný online v GitHub repozitári na adrese: `https://github.com/ViktorBojda/xmc`.

Pred začatím inštalácie je potrebné mať nainštalovaný Python vo verzií 3.11

Poznámka: Naše riešenie bolo testované na WSL2 s nainštalovaným operačným systémom Ubuntu 22.04

Podrobný postup inštalácie a spustenia:

1. Otvorte terminál, prejdite do koreňového priečinka so zdrojovým kódom a vytvorte virtuálne prostredie príkazom: `python3.11 -m venv .venv`
2. Aktivujte vytvorené prostredie pomocou príkazu: `source .venv/bin/activate`. Skontrolujte verziu Pythona príkazom: `python --version` (mala by byť 3.11).
3. Nainštalujte potrebné knižnice príkazom: `pip install -r requirements.lock`
4. Následne nainštalujte balík `shap` príkazom: `pip install .[shap]`
5. Spustite aplikáciu príkazom: `python -m xmc`
6. V konzole sa zobrazí zoznam dostupných klasifikátorov. Vyberte požadovaný klasifikátor zadaním príslušného čísla a potvrďte klávesom Enter.
7. Následne budete mať na výber z troch možností:
   - Zadajte **1** pre spustenie krížovej validácie
   - Zadajte **2** pre trénovanie a evaluáciu modelu
   - Zadajte **3** pre generovanie vysvetlení modelu
8. Ak si zvolíte možnosť **3** (generovanie vysvetlení), zobrazí sa ďalšie menu s výberom metódy vysvetľovania:
   - Zadajte **1** pre metódu SHAP
   - Zadajte **2** pre metódu Anchors
   - Zadajte **3** pre metódu kontrafaktuálne vysvetlenia