Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Learning Models on Data of Difficult Structure with Applications to Cybersecurity

Doctoral Thesis
Prague, February 2023

Ing. Tomáš Komárek
Supervisor: RNDr. Petr Somol, Ph.D.

Ph.D. Programme: Electrical Engineering and Information Technology (P2612)
Branch of study: Information Science and Computer Engineering (2612V025)

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Petr Somol, for his steady support in all my research initiatives since the very beginning of this very long Ph.D. journey. Petr has been a constant source of encouragement, providing insightful discussions and pointing me in the right directions to ensure the success of my research. I am incredibly grateful for the opportunity to work under his supervision and to learn from his expertise. My thanks also belong to his colleague, Tomáš Pevný, for providing invaluable insights and feedback that have been instrumental in shaping my research.

Next, I would like to thank my coworkers at Cisco Systems, Jan Brabec, Radek Starosta, and Štěpán Dvořák for their support, valuable feedback, and inspiring discussions. A special thanks also go to former colleagues Martin Grill, Lukáš Machlica, and Karel Bartoš for introducing me to the amazing world of cybersecurity and machine learning.

I am also grateful to Jan Kohout, my former colleague from Cisco Systems, and Jakub Lokoč and Přemysl Čech from the Faculty of Mathematics and Physics at Charles University in Prague for the great collaboration during our joint research work on exploring similarity search techniques for cybersecurity applications.

Last but not least, I would like to extend my sincere gratitude to my girlfriend, Jaja, for being my source of inspiration and for always standing by me. Her love and support have been a constant motivation and have made this journey much more enjoyable.

Abstract

A proper input data representation is essential for the successful application of learning models. While for homogeneous data like images, text, or sounds, end-to-end models capable of learning directly from raw data have already been developed, for heterogeneous data, such as JavaScript Object Notation (JSON) documents, manually designed feature representations are still the norm. Designing features by hand is, however, time-consuming and often suboptimal because not all of the discriminative information that a model could use is always extracted, even by domain experts. Moreover, this extra step in the machine learning pipeline might quickly become a bottleneck when experimenting with new data sources and/or in fast-evolving environments where the importance of individual features is subject to change and the whole process needs to be repeated again and again.

To address this issue, we propose a fully automated end-to-end approach for learning models on hierarchically structured data in JSON format, which eliminates the need for manual intervention. Collecting all the available information about each examined object into a single JSON file is all that is required. Then, depending on the task, a model will automatically decide which information is relevant and which can be discarded. To achieve this, we developed a JSON2bag tool that maps the problem of learning on JSONs into a Multiple Instance Learning (MIL) problem, as well as two independent MIL models that can be applied to general MIL problems: Bag-level Randomized Trees (BLRT) and Instance Selection Randomized Trees (ISRT).

On publicly available benchmark datasets, we demonstrate that each of the proposed methods outperforms prior-art solutions, even without careful hyper-parameter tuning. More importantly, we show on five real-world cybersecurity applications that MIL and learning on JSONs address a number of important problems in that field. Specifically, the flexibility of the JSON format makes it possible to combine weak signals of different types, sizes, and quality to improve the overall accuracy of predictions. Next, working with higher-level entities (represented by the JSONs) instead of the individual lower-level weak signals reduces label acquisition costs since there is less to annotate. Lastly, despite having labels only for high-level entities, the proposed method can naturally explain alerts by pointing out which low-level signals cause the alerts.

Keywords: end-to-end learning, JSON documents, tree-structured data, combining heterogeneous data sources, heterogeneous data types, learning with contextual information, multiple instance learning, decision tree ensembles, explainability, cybersecurity.

Abstrakt

Vhodná reprezentace vstupních dat je pro úspěšnou aplikaci učících se modelů klíčová. Zatímco pro homogenní data, jako jsou obrázky, text nebo zvuk, již byly vyvinuty end-to-end modely schopné učit se přímo z nezpracovaných dat, pro heterogenní data, jako jsou dokumenty JavaScript Object Notation (JSON), jsou stále standardem ručně navrhované příznakové reprezentace. Ruční navrhování příznaků je však časově náročné a často neoptimální, protože ne vždy se podaří vyextrahovat veškerou relevantní informaci, kterou by model mohl využít, a to ani expertům v dané oblasti. Navíc se tento dodatečný krok v procesu strojového učení může rychle stát úzkým hrdlem při experimentování s novými zdroji dat nebo v dynamicky se vyvíjejících prostředích, kde význam jednotlivých příznaků podléhá změnám a celý proces je potřeba neustále opakovat.

Jako řešení tohoto problému navrhujeme plně automatizovaný end-to-end způsob učení modelů nad hierarchicky strukturovanými daty ve formátu JSON, který eliminuje nutnost manuálního zásahu. Stačí pouze sesbírat veškeré dostupné informace o každém zkoumaném objektu do samostatného JSON souboru. Model pak v závislosti na konkrétní úloze automatizovaně rozhodne, které informace jsou relevantní a které lze vyřadit. K dosažení tohoto cíle jsme vyvinuli nástroj JSON2bag, který převádí problém učení nad soubory typu JSON na problém multi-instančního učení (MIL), a také dva samostatné MIL modely, které lze využít k řešení obecných MIL problémů: Bag-level Randomized Trees (BLRT) a Instance Selection Randomized Trees (ISRT).

Na veřejně dostupných referenčních datových sadách prokazujeme, že každá z navrhovaných metod překonává dosavadní známá řešení, a to i bez pečlivého ladění hyperparametrů. Co je však důležitější, na pěti reálných aplikacích v oblasti kybernetické bezpečnosti ukazujeme, že multi-instanční učení a učení nad soubory JSON řeší řadu důležitých problémů v této oblasti. Konkrétně flexibilita formátu JSON umožňuje kombinovat slabé signály různých typů, velikostí a kvality a zlepšit tak celkovou přesnost predikcí. Dále práce s entitami vyšší úrovně (reprezentovanými soubory JSON) namísto jednotlivých nízkoúrovňových slabých signálů redukuje náklady na anotaci dat, jelikož je díky agregaci méně entit k anotaci. A nakonec, přestože jsou k dispozici pouze anotace pro entity vyšší úrovně, navrhovaná metoda dokáže přirozeně vysvětlit pozitivní predikce poukázáním na signály nižší úrovně, které predikce způsobují.

Kličová slova: end-to-end učení, dokumenty JSON, stromově strukturovaná data, kombinování heterogenních zdrojů dat, heterogenní typy dat, učení s využitím kontextuální informace, multi-instanční učení, rozhodovací stromy, vysvětlitelnost, kybernetická bezpečnost.

Contents

Chapter 1

Introduction

Mathematical models play an essential role in numerous scientific and engineering applications as they help to understand and predict the behavior of complex natural and human-made systems. One type of mathematical model is a classification model, which assigns a given object into one of the predefined classes based on data representing the object. Even for the most basic form of two-class (binary) classification and one specific cybersecurity domain, there are tons of practical applications, such as distinguishing between spam and non-spam emails, malware and goodware files, fraudulent and legitimate transactions, forgery and genuine documents, criminal and regular customers, malicious and benign domains, malware-infected and clean network users, etc. In the current data-driven era, such classification models are typically constructed with supervised machine learning (ML) algorithms that extract discriminative patterns from provided datasets of sample pairs of objects and their respective classes. The final accuracy of the model depends on many factors, but it is known that the quality of data representation is among the most important ones, if not the most important [37].

ML algorithms usually request that every object be represented with a fixed set of numerical features, known as a feature vector. Since representation is critical to the success of ML, many years of research have gone into creating techniques for extracting relevant features from raw input data. Throughout the years, we have seen how human-defined features are being gradually outperformed by learned features derived automatically from raw data as part of the learning process. The most illustrative case is image analysis, where computer vision experts spent many years fine-tuning their definitions of various color-based and edge-based features [78, 30, 13], only to be radically overcome by Convolutional Neural Networks (CNN) [74]. Similar advances have also been made in modeling text with various word embedding techniques [81, 87] and transformer-based language models [95, 34, 21] or in the audio domain with speech-to-text deep neural network models [51, 11]. Arguably, methods in each of these domains benefit from (1) the regular nature of the data — image and video having a straightforward representation as tensors of constant dimensions, and audio and text having the form of a stream of numerical values of one type — and (2) the availability of massive datasets [33, 73, 111, 96, 85] — utilized for unsupervised [54] or self-supervised [36] model pretraining.

However, not every domain offers the same level of regularity in the data, nor the same abundance of large-scale datasets. For example, in the cybersecurity domain, the datasets are typically proprietary due to privacy issues, and the data have a *difficult structure*, posing a problem to existing learning techniques. To outline what we mean by a difficult structure, consider the task of classifying network users as either malware-infected or clean based on their connections with servers on the Internet. Since the number of contacted servers may vary from user to user (without any meaningful upper bound), it is rather difficult to represent users as fixed-size feature vectors. A small feature count may lead to information loss on users with many connections, while sufficiently large (if even possible) to over-parametrization and memory issues.

Moreover, each server may be described by yet another variable number of entities (e.g. users, files, domains) that are related to the particular server and are known to be malicious, leading to tree-structured data. Additionally, it might be important to model the entities as permutation-invariant, meaning that the order in which they appear should not affect the model's predictions. Furthermore, each entity type may have its own unique set of features, which can be of various data types, such as numerical, string, or Boolean values. Some features might also be missing occasionally. For instance, a domain reputation score feature may rely on an external service that could be temporarily unavailable or impose daily query limits. Finally, the whole data structure might not be fully under our control, and some sub-structures may change over time, requiring the model to exhibit robustness and/or fast retraining capabilities on modified data structures.

Since such hierarchically structured objects are commonly found in many other domains beyond cybersecurity, various data formats have been developed to represent them effectively. The JavaScript Object Notation (JSON) is currently among the most widely used [105]. It is a text-based, self-describing format that can store strings, numbers, Booleans, and null values in a tree-like structure of arbitrary depth and breadth, making it extremely versatile and useful for capturing many real-world phenomena. However, while JSONs are ubiquitous in the field of computer engineering and database management, they are rarely considered in the context of ML. This is because there is no universal method for fitting JSON data samples of variable tree structure into fixed-size feature vectors without significant information loss.

Therefore, objects of difficult structure are often represented in the JSON format for the convenience of storage, analysis, and human readability; but when ML is to be applied, a custom feature vector representation of the JSONs must be manually created for each particular task. However, explicit manual extraction of information is exactly the step that has been proven inferior to automatic extraction as part of the learning process. All mainstream large-scale models beat human-defined features. Moreover, hand-crafted features not only tend to be suboptimal, as humans struggle with identifying relevant information in complex data structures and are subject to bias, but the process is also costly and time-consuming, requiring domain experts and adding an extra bottleneck when experimenting with new data sources or adapting extractors to sudden changes in the JSON structure. Note that although relational learning [12] and particularly Graph Neural Networks (GNN) [114, 101] also address the problem of learning on structured data, they are not directly applicable to the JSON format and, therefore, do not make use of large amounts of already existing JSON datasets.

For this reason, we see a significant gap in the current state-of-the-art of ML in cybersecurity and beyond. The problem we aim to address in this thesis is to define scalable and easily operationalizable models that can utilize maximum information contained in JSON data common in cybersecurity — data usually collected from multiple sources of various quality, of a tree structure with unknown and potentially unstable schema, with a mix of various data types, a variable number of records, occasionally missing values and order-independent elements.

Unfortunately, the ability to model complex data structures does not necessarily resolve all practical issues when applying ML to fields like cybersecurity. Another critical problem we usually face is an *asymmetry in labeling requirements* with respect to the learning/application phase. A model must be able to learn using only incomplete non-specific high-level positive labels, but when applied, it must provide specific low-level explanations for positive high-level predictions. Let us clarify.

To recognize a malware infection, it is often necessary to analyze a combination of multiple data sources, including network traffic captures, operating system event logs, and file meta-data. Interpreting the data from any of these sources is, however, difficult and time-demanding, even for experts in the field. Moreover, the positive signal is often fragmented into numerous weaker ones that are easier to hide in a mass of background noise. Typically, the more important the signal is, the more attackers invest into hiding it. Due to these high labeling costs, and the ever-

increasing volume of new malware and data samples, it is practically impossible to compile a fully-labeled dataset that would be both representative and up-to-date. Supplying incomplete high-level labels (e.g. infected/clean user) without specifying the positive low-level signals (e.g. individual user's network communications responsible for the infection) is then one of the ways how at least significantly reduce the label acquisition costs.

On the other hand, in the application phase, it is generally not enough to just provide non-specific high-level decisions, because the alerts generated by the model cannot be blindly trusted and must be well justified. This is especially true when subsequent acting upon the raised alerts is associated with a high cost, such as reimaging a user's infected computer. In such cases, it is essential to provide explanations for the high-level positive predictions by identifying the specific low-level signals within the structured data that triggered the alarms. Given the potentially vast size of data structures, accurate identification of relevant low-level signals can greatly minimize the time human operators spend on validating predictions. This real-world need is often overlooked by traditional ML approaches, as the process of transforming complex data structures into fixed-size feature vectors renders the backward identification of positive signals within the original structures particularly challenging.

In this thesis, we will utilize the paradigm of Multiple Instance Learning (MIL) [5] to tackle both of the aforementioned key challenges: learning on data of difficult structure and dealing with the asymmetry in labeling requirements. MIL is an emerging branch of ML, where individual data samples (called bags in MIL terminology) are represented with multiple feature vectors (called instances) instead of single feature vectors (characteristics for the traditional so-called Single Instance Learning). Instances inside bags are independent of their order, and their count (i.e. bag size) may vary from bag to bag. Class labels are then available only for bags, not the inner instances.

Our strategy throughout the thesis is, therefore, to initially focus on developing universal MIL models that can effectively learn from training MIL datasets consisting of labeled bags to accurately classify new (previously unseen) bags. Since there is no supervision indicating which instances inside training bags are accountable for the bag label and which are irrelevant, the learning algorithms have to figure it out on their own. Then, by forcing the algorithm to explicitly select only the relevant instances and ignore the rest during the classification, we will address the issue of identifying low-level signals (i.e. instances) in positively classified high-level data structures (i.e. bags). After successfully tackling the problem of learning on bags, we will propose a transformation that maps the unsolved problem of learning on JSONs to the solved problem of learning on bags. Unlike the traditional ML approaches fitting arbitrary complex JSON samples into single fixed-size feature vectors, this dynamic-size multi-feature vector representation can be created with a minimum (to none) information loss and enable backward identification of low-level signals (specific JSON values) in positively classified high-level JSONs comprising many values.

1.1 Problem Statement

The key questions to be answered in this thesis can be summarized as follows:

1. Can we build MIL classifiers on top of flexible data structures where objects are represented as bags of instances that would outperform prior-art MIL methods in accuracy, simplicity (single-step methods), and robustness (performing well over a broad range of problems without extensive hyper-parameter tuning)?
2. Can we build MIL classifiers that can *explicitly* identify the subset of relevant instances within positively classified bags (and effectively prioritize the instances within the subset), so that we minimize the time of human operators verifying the findings?

3. Can we build MIL classifiers on top of JSONs in an end-to-end fashion with an automated JSON-to-bag conversion that extracts maximum available information from every JSON without relying on expensive human-defined *task-specific* feature extraction, nor the availability or even existence of common JSON schema?
4. Can we build MIL classifiers on top of JSONs that are robust against typical JSON distortions such as schema drift, pollution with noisy values, and positive signal displacement?
5. Can we build MIL classifiers that outperform traditional ML techniques as well as specialized prior-art MIL-based, JSON-based, and GNN-based methods on several real-world cybersecurity MIL/JSON datasets?

1.2 Proposed Solution

Addressing the above-defined problems requires the construction of powerful MIL classification models. In this thesis, we propose two independent algorithms, Bag-level Randomized Trees (BLRT) and Instance Selection Randomized Trees (ISRT), as potential solutions for deriving such models from datasets of labeled bags.

Both algorithms, as their names suggest, are based on ensembles of (non-boosted) decision trees, which are still widely used in the cybersecurity industry and other fields, even in this neural network age. The reason for this is easy scalability to large datasets [1] (individual trees can be built in parallel, independently), and usually good, if not excellent, out-of-the-box performance that can be achieved over a wider range of problems without the need for problem-specific hyper-parameter tuning or data preprocessing (e.g. feature rescaling) [42][1][2]. One of the key factors responsible for the robust performance of decision trees is the built-in feature selection mechanism that robustly seeks the locally optimal feature-threshold split for each node in the tree during the learning process. Not only it decreases the complexity of a model and the risk of overfitting by excluding irrelevant and noisy features, but it can also provide feature importance rankings, which might help to understand the decision-making process. As such, our goal within the BLRT and ISRT algorithms is to adjust the traditional tree node-splitting mechanism to the MIL setting, where samples (i.e. bags) may consist of multiple feature vectors (i.e. instances). Particularly, within ISRT, we further aim to extend the feature-selection mechanism onto instances (therefore the name Instance Selection Randomized Trees) and thus recognize which instances inside bags are important for the decision.

Learning on JSONs is then approached by mapping individual JSON objects to MIL bags. Since JSONs possess a variable breadth and depth tree structure, and MIL bags provide only a one-level dynamic structure, we propose a tool called JSON2bag that disassembles the JSON tree structure into branches and represents each branch as a single instance within a bag. The objective of the JSON2bag mapping is to fully harvest the information contained in JSONs to bags so that MIL models can decide themselves what information is important and what is not in relation to a given task, without relying on human judgment. Converting all JSON values into bag instances, without imposing any count limitation, also paves the way for the ISRT model to retrospectively pinpoint the relevant ones.

[1] The consistently good performance of models, such as Breiman's Random Forests [18], over a wide range of conditions, presents a distinct advantage in practical ML applications. The robustness, especially when the model is periodically retrained with new data, mitigates the sudden risk of producing inaccurate models (e.g. due to concept drift in the new data [57]) that could generate an excessive number of false alarms, thereby making the entire security system useless.

[2] On the other hand, unlike neural networks, decision tree-based algorithms cannot create pre-trained models that can be later reused as building blocks for various other tasks, as exemplified by https://huggingface.co/ project. In other words, each time a new problem needs to be solved, a new model must be trained from scratch.

1.3 Contributions

This thesis has four main contributions:

1. **Bag-level Randomized Trees (BLRT)** (Chapter 3) We propose a novel classification model as an extension of traditional (single instance) decision trees to the MIL setting, where samples (called bags) are represented as sets of feature vectors (called instances). To be able to decide in every tree node, whether a bag should continue to the left or right branch, during the propagation from the root node to the leaves, we extend the standard decision tree condition evaluating — if a value of a specific feature is greater than a certain threshold — by an additional parameter judging the percentage of instances within the bag that accomplish the condition. The added parameter is learned for each node separately along with the standard feature index and threshold value parameters during the tree construction process. Surprisingly, despite conceptual simplicity, the BLRT model is demonstrated to outperform 28 prior-art MIL classifiers on 29 publicly accessible benchmark datasets, even without hyper-parameter tuning.
2. **Instance Selection Randomized Trees (ISRT)** (Chapter 4) is another MIL classifier proposed in this thesis which addresses one additional requirement beyond the capabilities of BLRT — regularizing the model to minimize the number of instances that the model selects for building its decision boundary. This requirement is motivated by the real needs of cybersecurity systems where analyzed entities are typically described by large amounts of complex data that are difficult and costly to interpret by humans, but at the same time, the verdicts upon positively classified entities must be well justified using the data. Therefore we need to train classifiers that minimize the amount of information based on which decisions are made and thus to save time of human operators post-processing such decisions. To achieve that we implement an instance-selection mechanism explicitly selecting only the most informative instance in each tree node upon which the standard feature-threshold condition is evaluated. On benchmark datasets, we demonstrate that ISRT indeed exhibits a strong instance reduction effect without sacrificing accuracy. We also show that the number of times an instance is selected reflects its importance and that the introduced regularization may also lead to higher accuracy by eliminating the influence of noisy and irrelevant instances from the decision-making process.
3. **JSON2bag** (Chapter 5) is an automated feature extraction algorithm that we propose for converting JSON documents to MIL bags, thereby enabling learning on top of JSONs. Given an arbitrary complex JSON sample, JSON2bag decomposes its JSON tree topology into a set (i.e. bag) of instances, where each instance represents a single JSON value along with its location within the tree topology. Unlike the closest prior-art method, JSON2bag does not assume the existence of a schema that all JSONs from a dataset must obey. This allows us to model even schema-less problems, which we illustrate in classifying web pages, where every page may have a different structure of the HTML source code. We also examine the robustness of the proposed representation against two types of common JSON distortions: pollution with irrelevant noisy values and signal displacement (i.e. a move of JSON subtree to another position within the same tree in future samples). Finally, we show that, while JSON2bag is compatible with any MIL model, in conjunction with ISRT, it can explain decisions by means of the top most important JSON values.
4. **Applications to cybersecurity** (Chapter 6) We demonstrate the effectiveness of our proposed methods by comparing them to both traditional and specialized prior-art ML approaches in five real-world cybersecurity applications. The data used are authentic and their structure has not been altered in any way to ensure the closest possible resemblance to real-world conditions. We also place a particular emphasis on evaluation protocols to ensure that any improvements are not simply due to overfitting.

1.4 Publications

This thesis builds on and extends the work presented in the following publications:

1. **Tomáš Komárek**, Jan Brabec and Petr Somol. Explainable Multiple Instance Learning with Instance Selection Randomized Trees. *Conference: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD), 2021* [64] [CORE A].
2. **Tomáš Komárek** and Petr Somol. Multiple Instance Learning with Bag-level Randomized Trees. *Conference: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD), 2018* [67] [CORE A].

Other author's publications not covered by the thesis:

3. **Tomáš Komárek**, Jan Brabec, Čeněk Škarda and Petr Somol. Threat Hunting as a Similarity Search Problem on Multi-positive and Unlabeled Data. *Conference: IEEE International Conference on Big Data (BigData), 2021* [62] [CORE B].
4. Jan Brabec, **Tomáš Komárek**, Vojtěch Franc and Lukáš Machlica. On Model Evaluation Under Non-constant Class Imbalance. *Conference: International Conference on Computational Science (ICCS), 2020* [17] [CORE A].
5. Jan Kohout, **Tomáš Komárek**, Přemysl Čech, Jan Bodnar and Jakub Lokoč. Learning Communication Patterns for Malware Discovery in HTTPs Data. *Journal: Expert Systems with Applications, 2018* [60] [Impact factor 8.665].
6. **Tomáš Komárek** and Petr Somol. End-node Fingerprinting for Malware Detection on HTTPS Data. *Conference: International Conference on Availability, Reliability and Security (ARES), 2017* [66] [CORE B]. Also published as *US Patent: 10,867,036, 2020* [68].
7. Přemysl Čech, Jan Kohout, Jakub Lokoč, **Tomáš Komárek**, Jakub Maroušek and Tomáš Pevný. Feature Extraction and Malware Detection on Large HTTPS Data using MapReduce. *Conference: International Conference on Similarity Search and Applications (SISAP), 2016* [121] [CORE B].
8. **Tomáš Komárek**, Martin Grill and Tomáš Pevný. Passive NAT Detection using HTTP Access Logs. *Conference: IEEE International Workshop on Information Forensics and Security (WIFS), 2016* [63]. Also published as *US Patent: 9,667,636, 2017* [65].
9. **Tomáš Komárek** and Petr Somol. Instant Network Threat Detection System. *US Patent: 11,374,944, 2022* [70].
10. **Tomáš Komárek** and Petr Somol. Generating a Vector Representative of User Behavior in a Network. *US Patent: 11,271,954, 2022* [69].
11. **Tomáš Komárek**, Martin Vejman and Petr Somol. Training a Network Traffic Classifier Using Training Data Enriched with Contextual Bag Information. *US Patent: 11,271,833, 2022* [71].
12. Radovan Haluška, Jan Brabec and **Tomáš Komárek**. Benchmark of Data Preprocessing Methods for Imbalanced Classification. *Conference: IEEE International Conference on Big Data (BigData), 2022* [50] [CORE B].

Tools developed in this research are publicly available at:

- https://github.com/komartom/BLRT.jl
- https://github.com/komartom/ISRT.jl
- https://github.com/komartom/JSON2bag.jl

Chapter 2

Background and Related Work

In the previous chapter, we defined and motivated the problem of learning on data of difficult structure, including the problem of asymmetrical labeling requirements w.r.t. the learning/application phase. In this chapter, we introduce background preliminaries and review related work in the fields of Multiple Instance Learning (MIL) and learning on JavaScript Object Notation (JSON) documents, which are both relevant for addressing the stated problems.

2.1 Multiple Instance Learning

Multiple Instance Learning (MIL) [5] generalizes the traditional data representation as it allows individual data samples $\mathcal{B}_1, \mathcal{B}_2, \ldots$ (called bags) to be represented as a set of multiple d-dimensional feature vectors $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$, $\mathbf{x} \in \mathbb{R}^d$ (called instances), which are order independent and their counts may vary across bags. In the supervised classification, it is further assumed that each bag is associated with label y (e.g. $y \in \{-1, +1\}$ in the binary case), and the goal is to infer function $\mathcal{F}$ from dataset $\mathcal{D} = \{(\mathcal{B}, y)_1, (\mathcal{B}, y)_2, \ldots\}$, using algorithm $\mathcal{A}$, such that the function $\mathcal{F}$ can predict labels for new (previously unseen) bags $\mathcal{F}(\mathcal{B}) = y$.

The definition of MIL goes back to 1997 [35]. One of the first examples illustrating the concept of MIL is the keychain problem [9]: Imagine there is a locked door and a collection of peoples' keychains (bags), each consisting of several keys (instances). We know about certain individuals who can or cannot open the door (labeled bags), but we do *not* know which key on their keychains makes this possible (unlabeled instances). Based on these keychains, our goal is to learn

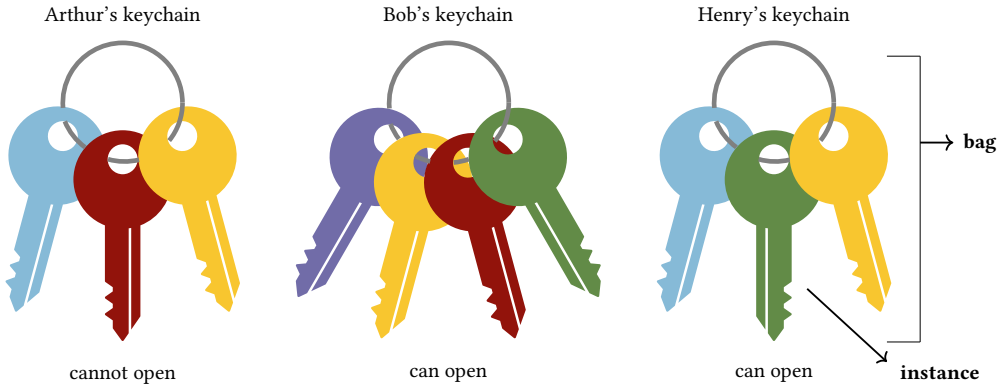

Figure 2.1: Multiple Instance Learning (MIL) illustrated on keychain problem. Unlike traditional learning, the task is to learn from high-level, labeled bags (keychains) rather than low-level instances (keys). A model should learn to classify keychains as 'can open' if contain a green key.

to identify who else's keychain can open the door (bag classifier $\mathcal{F}$). In the context of Figure 2.1, the correct inference would be that any keychain with the green key can unlock the door.

While MIL's primary objective is to predict labels for high-level bags (Can a given **keychain** unlock the door?), in some applications, it is more important to identify the low-level "positive" instances that cause positive bag labels (Can a given **key** unlock the door?). As such, MIL formalism is typically utilized to address one of these two seemingly distinct problems:

- **Learning on variable-sized data**: The aim is to classify objects of variable size (e.g. keychains) that can be represented as bags - that is, objects divisible into order-independent components, or instances, which are characterized using the same set of features, but their count may vary from object to object.
- **Learning on weakly labeled data**: In this scenario, the objects of interest are the instances themselves (e.g. keys). The aim is to construct an instance-level classifier $\mathrm{f}(\mathbf{x}) \rightarrow \{-1, +1\}$ using the bag-level labels only, thus reducing the number of labels required for training. Interestingly, as we will see in the next Section 2.1.1, this can be viewed as a special case of the previous problem, assuming that the discriminative information lies solely on the level of instances and that there is an explicit relation between bag and instance labels, e.g., $\mathcal{F}(\mathcal{B}) = \max_{\mathbf{x} \in \mathcal{B}} \mathrm{f}(\mathbf{x})$.

Although MIL can still be seen as an emerging branch of Machine Learning, there is already a decent amount of prior art available [9, 23, 53, 119]. Amores [5] proposes a taxonomy of MIL algorithms, categorizing them based on whether they extract discriminative information from local, *instance-level* (Instance-space paradigm 2.1.1) or global, *bag-level*; and in the later case whether they operate in the original non-vectorial bag-space (Bag-space paradigm 2.1.2) or map non-vectorial bags into a vectorial embedded space (Embedded-space paradigm 2.1.3).

2.1.1 Instance-space Paradigm

In the Instance-space paradigm, the discriminative information is considered to lie at the level of instances. In other words, to determine a bag's label, it is enough to classify its instances one by one in isolation and aggregate the verdicts. This holds, for example, for the keychain problem as outlined in Figure 2.1, where recognizing the green key suffices for the entire keychain to be labeled as "can open". Methods based on this paradigm, therefore, aim to infer an instance-level classifier:

$$\mathrm{f}(\mathbf{x}) \rightarrow \{-1, +1\} \tag{2.1}$$

from a dataset of labeled bags $\mathcal{D} = \{(\mathcal{B}, y)_1, (\mathcal{B}, y)_2, \ldots\}$, where $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\},\ \mathbf{x} \in \mathbb{R}^d$. Since the labels $y \in \{-1, +1\}$ are assigned to bags of instances instead of individual instances, an assumption must be made about their relationship to instances, in order to solve this problem. The most commonly used assumption is referred to as the standard multi-instance assumption.

Standard multi-instance assumption: *In every positive bag, there is at least one positive instance (i.e. an instance carrying a discriminative positive signal), while in negative bags there is no such instance — all of the instances are negative (i.e. not carrying the positive signal).*

By denoting instance labels as $y_{\mathbf{x}} \in \{-1, +1\}$, the assumption can be formalized as follows:

$$y = \max_{\mathbf{x} \in \mathcal{B}} y_{\mathbf{x}}, \qquad \mathcal{F}(\mathcal{B}) = \max_{\mathbf{x} \in \mathcal{B}} \mathrm{f}(\mathbf{x}), \tag{2.2}$$

where the max rule ensures that a bag $\mathcal{B}$ is positive if at least one of its instances $\mathbf{x}$ is positive.

The fact that one instance with some desirable properties makes the whole bag positive naturally occurs in many real-world situations. For example, in the drug activity prediction problem [35], each classified chemical molecule can take multiple shapes — represented as a bag of instances. If a molecule can take a shape that strongly binds to a target protein, then it is an effective drug. Otherwise, the drug is ineffective.

Another example is the detection of malware-infected devices [107], where each device, storing multiple files, is represented as a bag of instances. If any file within a device is classified as malicious, then the entire device is deemed infected (and might be isolated from the network). Based on this assumption, MIL can vastly reduce labeling costs by enabling the creation of standard file-centric anti-virus models (capable of explicitly identifying malicious files) using only device-level labels, effectively avoiding the need to annotate every single file in the training dataset. Moreover, to obtain a positive device-level label, it is enough to observe just a manifestation of malicious behavior like excessive pop-up advertisements, unwanted redirects to unfamiliar websites, intrusive toolbars, screen lockdowns, ransomware payment demands, etc. that might not be necessarily file-related.

One of the first instance-space algorithms following the standard assumption is the Axis-Parallel-Rectangle (APR) method [35]. To identify the type of instances that cause the positive bag label, the method builds an instance-level classifier $\mathrm{f}(\mathbf{x}; \mathcal{R})$ of the form:

$$\mathrm{f}(\mathbf{x}; \mathcal{R}) = \begin{cases} +1, & \text{if } \mathbf{x} \in \mathcal{R} \\ -1, & \text{otherwise,} \end{cases} \tag{2.3}$$

where $\mathcal{R}$ denotes an axis-parallel hyper-rectangle in the instance space. By shrinking and growing, the parameter $\mathcal{R}$ is optimized to simultaneously maximize the number of positive bags in the training set that contain at least one instance in $\mathcal{R}$ and minimize the number of negative bags that contain any instance in $\mathcal{R}$.

Yang *et al.* [115] propose Asymmetrical-SVM (ASVM), an extension of the traditional Support Vector Machines (SVM) to the MIL setting under the standard assumption. ASVM converts a MIL problem to the conventional single-instance learning problem, by letting all instances $\mathbf{x}_i$ inherit their bag label $\widetilde{y}_{\mathbf{x}_i}$ and searching for a separating hyperplane $(\mathbf{w}, b)$ that maximizes margins between the positive and negative instances subject to the introduced asymmetrical loss function. ASVM assigns a higher misclassification cost to false positives than false negatives, because a false positive instance in a negative bag always means an error in the bag label prediction, while a false negative instance in a positive bag does not necessarily result in an error if there are further positive instances in the bag. The optimization problem is defined as:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & \widetilde{y}_{\mathbf{x}_i}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \frac{\widetilde{y}_{\mathbf{x}_i} + 1}{2} \cdot \xi_i \\ & \xi_i \geq 0 \,, \end{aligned} \tag{2.4}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the normal vector of the separating hyperplane, $b \in \mathbb{R}$ is the hyperplane offset, $\xi_i \geq 0$ are slack variables allowing classification errors, and $C > 0$ is a hyper-parameter balancing the bias-variance trade-off between fitting the training data and the generalization capability. The introduced term $\frac{\widetilde{y}_{\mathbf{x}_i}+1}{2}$ ensures that all negative instances are on the negative side of the hyperplane (no false positives) while minimizing false negative predictions.

Andres *et al.* [7] present MI-SVM, a MIL variant of SVM that directly incorporates the max rule from the standard assumption (Equation 2.2) into the loss function. The objective is to avoid forcing SVM to assign a positive value to all the instances of a positive bag, but only to at least one of the instances:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{\mathcal{B}} \xi_{\mathcal{B}} \\ \text{s.t.} \quad & y \cdot \max_{\mathbf{x} \in \mathcal{B}} (\langle \mathbf{w}, \mathbf{x} \rangle + b) \geq 1 - \xi_{\mathcal{B}} \\ & \xi_{\mathcal{B}} \geq 0 \,. \end{aligned} \tag{2.5}$$

It is a bag-centered formulation, where only one instance per bag matters. For a positive bag, the margin is defined by the most positive instance, while for a negative bag, the margin is defined by the least negative instance. Unlike ASVM, MI-SVM is a non-convex problem, making it challenging to solve using traditional optimization methods. Andres *et al.* [7] cast the problem to a mixed-integer programming problem and suggest optimization heuristics.

Zucker and Chevalier [29] introduce ID3-MI, the first adaptation of the popular Iterative Dichotomiser 3 (ID3) algorithm [94] for building decision trees to the MIL setting. ID3 uses a top-down, greedy strategy to construct a tree structure model $\mathrm{f}(\mathbf{x})$. It recursively divides training instances into subsets based on attributes that yield the highest information gain at each step. This process continues until either a pure classification is achieved or no more attributes are available. ID3-MI then uses a redefined version of the information gain measure accounting for the fact that multiple instances might belong to one bag:

$$\begin{aligned}\mathrm{Entropy}(\mathcal{S}) = \; & -\frac{\mathrm{P}(\mathcal{S})}{\mathrm{P}(\mathcal{S})+\mathrm{N}(\mathcal{S})}\log_2\left(\frac{\mathrm{P}(\mathcal{S})}{\mathrm{P}(\mathcal{S})+\mathrm{N}(\mathcal{S})}\right) \\ & -\frac{\mathrm{N}(\mathcal{S})}{\mathrm{P}(\mathcal{S})+\mathrm{N}(\mathcal{S})}\log_2\left(\frac{\mathrm{N}(\mathcal{S})}{\mathrm{P}(\mathcal{S})+\mathrm{N}(\mathcal{S})}\right)\end{aligned} \tag{2.6}$$

$$\mathrm{InfoGain}(\mathcal{S}, A) = \mathrm{Entropy}(\mathcal{S}) - \sum_{v\in\mathrm{Values}(A)} \frac{\mathrm{P}(\mathcal{S}_v)+\mathrm{N}(\mathcal{S}_v)}{\mathrm{P}(\mathcal{S})+\mathrm{N}(\mathcal{S})} \cdot \mathrm{Entropy}(\mathcal{S}_v), \tag{2.7}$$

where $\mathrm{P}(\mathcal{S})$ and $\mathrm{N}(\mathcal{S})$ denote, respectively, the number of positive and negative *bags* having instances in subset $\mathcal{S}$. Symbol A stands for an attribute partitioning the subset $\mathcal{S}$ into $\{\mathcal{S}_1, \ldots, \mathcal{S}_v\}$ based on its values $v \in \mathrm{Values}(A)$. Since instances of a bag may be dispersed across various branches of the tree, in the prediction phase, a bag is classified as positive if at least of its instances reaches a positive leaf node.

Blockeel *et al.* [15] propose Multi-Instance Tree Inducer (MITI), an improved version of ID3-MI that implements heuristics to address issues related to the dispersion of instances. MITI grows a tree in the best-first order, prioritizing the expansion of nodes with instances belonging to a higher number of positive bags. A priority queue of nodes that have not yet been expanded is maintained. When the head node contains instances of positive bags only, it is transformed into a leaf and all instances of bags associated with this leaf are removed from the remaining nodes in the queue. The second heuristic is the use of a weighted Gini impurity measure [19] to give more weight to instances of smaller bags.

Leistner *et al.* [75] present MIForest, an ensemble of multiple-instance decision trees inspired by the Random Forest algorithm [18]. Ensembles of trees are typically more accurate than individual trees, as they reduce the characteristic high variance of individual decision trees by combining their predictions. To ensure that the trees within an ensemble are different from one another, Random Forest injects randomness into their growing process by (1) training each tree on a randomly resampled dataset (sampling with replacement) and (2) by considering only a random subset of features for selection at each split. MIForest then follows the standard multi-instance assumption (Equation 2.2) and considers labels of instances in positive bags as random variables defined over a space of probability distributions. To find their true but hidden labels, MIForest searches for distributions that minimize the overall learning objective. Since this is a non-convex optimization problem, a deterministic annealing technique [98] is employed to solve it iteratively.

Straehle *et al.* [108] propose MIOForest, a multiple-instance response-optimized Random Forest. MIOForest builds upon and extends the previous MITI algorithm [15] in four aspects. First, a forest is trained instead of a single tree. Second, the traditional axis-orthogonal splitting rules are replaced with non-linear ones to consider multiple features at a time. Third, mechanisms for tree regularization and instance weight redistribution are implemented to increase robustness

against noise in positive bags. Finally, an optimal combination of decision outputs of all trees in the forest is learned under the standard assumption.

Collective MI assumption: *Every instance in a bag contributes **equally** to the bag's label.*

Consider the following redefinition of the keychain problem to illustrate the situation in which the standard assumption is no longer appropriate while the collective assumption is. Imagine that you have to blindly select one key from a keychain to open a door, and you only have one try. Then, a positively labeled keychain is one with a high probability (let's say $\geq 90\%$) that a randomly selected key will open the door. In this scenario, each key on a keychain contributes *equally* to the ratio of (positive) green keys, which can open the door, and (negative) non-green keys, which cannot. This ratio then determines the keychain's final label, indicating the likelihood of picking a successful key. In other words, it is necessary to consider each instance-level classification equally to determine the final bag's label correctly. Note that deciding based on the first positively classified instance (the standard assumption) would produce false positive predictions on keychains with fewer green keys than non-green ones.

Frank and Xu [46] present a Wrapper MI algorithm for estimating the instance-level classifier $f(\mathbf{x})$ under the collective assumption. It simply trains a standard supervised classifier on a training set of instances where each instance inherits the label of its parent bag. Before training, the instances are weighted so that each bag receives the same total weight in the training. Given a new bag $\mathcal{B}$, the bag-level classifier $\mathcal{F}(\mathcal{B})$ is obtained using the sum as the aggregation rule:

$$\mathcal{F}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathrm{f}(\mathbf{x}). \tag{2.8}$$

Foulds [44] further generalizes the collective assumption to a weighted collective assumption and specifies a weight function over the instance space. This function $w(\mathbf{x})$ assigns weights to individual instances, enabling bag-level predictions to be calculated as the weighted sum of instance-level responses: $\mathcal{F}(\mathcal{B}) = \frac{1}{\sum_{\mathbf{x} \in \mathcal{B}} w(\mathbf{x})} \sum_{\mathbf{x} \in \mathcal{B}} w(\mathbf{x}) \mathrm{f}(\mathbf{x})$. The assumption states that each instance contributes to the bag's label independently, but not necessarily equally. It provides a smooth transition between the standard and the collective assumption. If zero weights are assigned to all negative instances and non-zero weights to positive instances, we achieve the standard assumption. And, if an equal weight is assigned to all instances, we end up with the collective assumption.

2.1.2 Bag-space Paradigm

In the Bag-space paradigm, the discriminative information is considered to lie at the level of bags. In other words, to determine a bag's label, the bag is classified directly as a whole entity $\mathcal{F}(\mathcal{B}) \rightarrow \{-1, +1\}$ based on *global* bag-level information extracted from possibly all bag's instances, instead of classifying individual instances one by one based on *local* instance-level information $\mathrm{f}(\mathbf{x})$ and aggregating the verdicts $\mathcal{F}(\mathcal{B}) = \mathrm{agg}_{\mathbf{x} \in \mathcal{B}} \mathrm{f}(\mathbf{x})$ (i.e. Instance-space paradigm 2.1.1). Hence, the assumption that there are labels at the instance level is no longer held, and only a bag-level classifier $\mathcal{F}$ is inferred. This allows for more informed decisions about the class of $\mathcal{B}$.

As an example of when local instance-level information is insufficient and global bag-level must be used, consider the redefined keychain problem in Figure 2.2. Now, two different locks, green and blue, secure the door. For access, you need a keychain with both a green key and a blue key on it. If you have just one key or none of them on the keychain, you won't be able to open the door. Put into MIL terminology, there are two types of "positive" instances (keys) that must co-occur in a bag (keychains) to be labeled as positive. A negative bag, on the other hand, does not contain any such instance, or either just instance(s) of the first type (a green key) or the second type (a blue key), but not both at the same time. Therefore, aggregating instance-level decisions is not enough as it would lead to false positives when there is only one type of

"positive" instance in a bag. To prevent this, we must extract the global bag-level information, taking into account the composition of the entire bag, prior to making the class prediction.

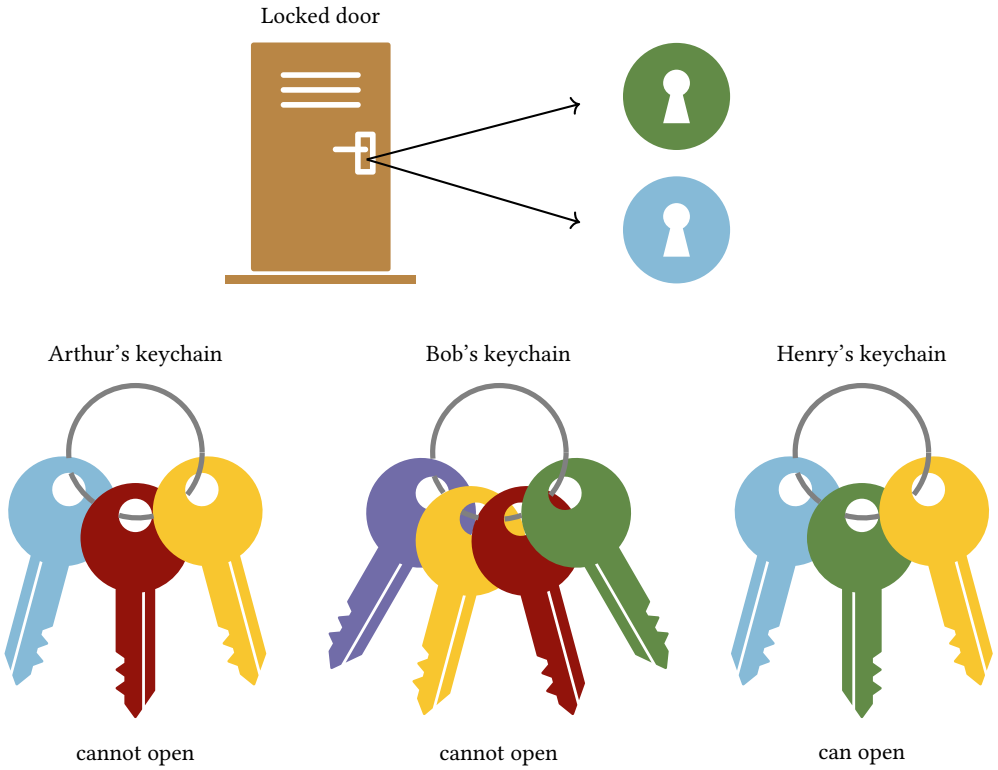

Figure 2.2: Redefinition of the MIL keychain problem, emphasizing the necessity of extracting *global* bag-level information over *local* instance-level one. At this time, the door is secured by two distinct locks. To open it, one requires a keychain that holds both the green and blue keys. Having neither or only one of these two keys on the keychain means the door cannot be opened. Therefore, aggregating instance-level decisions (Instances-space paradigm 2.1.1) is insufficient as it would lead to false positive predictions in the case of Arthur's and Bob's keychains.

The common strategy for classifying non-vectorial entities, like bags (i.e. sets of d-dimensional vectors $\mathbf{x} \in \mathbb{R}^d$), is defining a distance function $\mathrm{D}(\mathcal{B}_i, \mathcal{B}_j) \to \mathbb{R}_{\geq 0}$ that compares any two bags $\mathcal{B}_i$ and $\mathcal{B}_j$, and integrating it into a conventional distance-based classifier, such as the K-Nearest Neighbor (K-NN) or SVM algorithm. The bag-to-bag distance function $\mathrm{D}(\mathcal{B}_i, \mathcal{B}_j)$ is usually defined as an aggregation of distances between each bag's instances $\mathrm{d}(\mathbf{x}^i, \mathbf{x}^j) \to \mathbb{R}_{\geq 0}$. While the Euclidean distance $d(\mathbf{x}^i, \mathbf{x}^j) = \sqrt{\sum_{f=1}^{d}(x_f^i - x_f^j)^2}$ is commonly used for this purpose, alternative measures like Manhattan, Minkowski, and Chebyshev distances can be utilized as well [24].

Minimal Hausdorff distance [112] is one example of such distance function D. It is defined as the minimal distance between an instance from $\mathcal{B}_i$ and an instance from $\mathcal{B}_j$:

$$\mathrm{D}^{\mathrm{Min}}_{\mathrm{Hausdorff}}(\mathcal{B}_i, \mathcal{B}_j) = \min_{\mathbf{x}^i \in \mathcal{B}_i} \min_{\mathbf{x}^j \in \mathcal{B}_j} \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j). \tag{2.9}$$

The Minimal Hausdorff distance might be problematic in many contexts, as it focuses only on the closest instances between the bags and ignores the rest of the information contained in the bags. Also, it is not a metric as it does not satisfy the identity property, meaning a zero distance does not imply $\mathcal{B}_i = \mathcal{B}_j$.

Maximal Hausdorff distance [112] is another typical measure. It captures the maximum of all the distances from an instance in bag $\mathcal{B}_i$ to the closest instance in bag $\mathcal{B}_j$ and vice versa:

$$\mathrm{D}^{\mathrm{Max}}_{\mathrm{Hausdorff}}(\mathcal{B}_i, \mathcal{B}_j) = \max\left\{ \max_{\mathbf{x}^i \in \mathcal{B}_i} \min_{\mathbf{x}^j \in \mathcal{B}_j} \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j), \max_{\mathbf{x}^j \in \mathcal{B}_j} \min_{\mathbf{x}^i \in \mathcal{B}_i} \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j) \right\}. \tag{2.10}$$

For each instance, the closest instance in the other bag is located and the maximum of these distances is used as the bag distance. Unlike the minimal Hausdorff distance, this distance is a metric as it satisfies the identity, symmetry, and triangle inequality properties. However, it is known to be sensitive to outliers.

Chamfer distance [40] is a measure developed for digital geometry, specifically for point cloud modeling where sets of points (i.e. bags of 2D/3D-dimensional instances) constitute digital shapes in images. It is computed as the average distance between an instance in $\mathcal{B}_i$ and its closest instance in $\mathcal{B}_j$ and vice versa:

$$\mathrm{D}_{\mathrm{Chamfer}}(\mathcal{B}_i, \mathcal{B}_j) = \frac{1}{|\mathcal{B}_i|} \sum_{\mathbf{x}^i \in \mathcal{B}_i} \min_{\mathbf{x}^j \in \mathcal{B}_j} \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j) + \frac{1}{|\mathcal{B}_j|} \sum_{\mathbf{x}^j \in \mathcal{B}_j} \min_{\mathbf{x}^i \in \mathcal{B}_i} \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j). \tag{2.11}$$

For each instance from one bag, the nearest neighbor instance in the other bag is found, and the computed distances are summed up. The Chamfer distance is low if two bags $\mathcal{B}_i$ and $\mathcal{B}_j$ have the same or similar types of instances. In contrast to the Hausdorff distance, it is less sensitive to noise and outliers.

Earth Mover's Distance (EMD) [117] is another popular measure that was originally developed for content-based image retrieval. EMD can be interpreted as a metric between two multivariate probability distributions that are represented as bags of instances within a given instance space. The computed distance corresponds to the minimal cost that must be paid to transform (move) one distribution (pile of the earth) into the other (hole in the ground) in a defined sense:

$$\mathrm{D}_{\mathrm{EMD}}(\mathcal{B}_i, \mathcal{B}_j) = \frac{\sum_{\mathbf{x}^i \in \mathcal{B}_i} \sum_{\mathbf{x}^j \in \mathcal{B}_j} w_{ij}\ \mathrm{d}(\mathbf{x}^i, \mathbf{x}^j)}{\sum_{\mathbf{x}^i \in \mathcal{B}_i} \sum_{\mathbf{x}^j \in \mathcal{B}_j}\ w_{ij}}, \tag{2.12}$$

where the weights $w_i j$ are obtained through an optimization process that, given specified restrictions [99], aims to globally minimize $\mathrm{D}(\mathcal{B}_i, \mathcal{B}_j)$. While the calculation of EMD is resource-intensive due to the involved optimization, Amores [5] demonstrates its effectiveness on 11 MIL datasets, where the EMD+SVM approach significantly outperforms all four representative methods of the Instance-space paradigm and yields consistently good results.

2.1.3 Embedded-space Paradigm

In the Embedded-space paradigm, the discriminative information is also considered to lie at the global bag level, similar to the previous Bag-space paradigm 2.1.2. However, unlike the latter where methods operate in the original non-vectorial bag space, methods following the Embedded-space paradigm transform the non-vectorial bags into a vectorial "embedded" space, where the traditional single-instance learning is carried out. In other words, embedded-space approaches, rely on a mapping $\mathbf{\Phi}(\mathcal{B}) \rightarrow \mathbb{R}^D$ that summarizes global bag-level information from multiple instances of bag $\mathcal{B}$ into a single D-dimensional feature vector, allowing the bag-level classifier $\mathcal{F}$ to be defined as a conventional single-instance model f:

$$\mathcal{F}(\mathcal{B}) = \mathrm{f}(\mathbf{\Phi}(\mathcal{B})). \tag{2.13}$$

Given a new bag $\mathcal{B}$ to be classified with $\mathcal{F}(\mathcal{B}) \rightarrow \{-1, +1\}$, we first map the bag into a single feature vector representation using the mapping $\mathbf{\Phi}(\mathcal{B})$, and then apply the classifier f. The mapping function $\mathbf{\Phi}$ can be constructed in a number of different ways, each focusing on different

types of information. The choice of the way can significantly influence the effectiveness of this approach.

Dong [38] uses a straightforward mean aggregation, calculated for each feature $f \in \{1, \ldots, d\}$ across all instances in a bag, as the mapping function. Following this, Gärtner *et al.* [49] propose a Min-Max kernel. This mapping employs both the minimum and maximum aggregation and concatenates them into one feature vector that is twice the length of the instance dimension:

$$\boldsymbol{\Phi}(\mathcal{B}) = (\min_{\mathbf{x}\in\mathcal{B}} x_1, \ldots, \min_{\mathbf{x}\in\mathcal{B}} x_d, \max_{\mathbf{x}\in\mathcal{B}} x_1, \ldots, \max_{\mathbf{x}\in\mathcal{B}} x_d). \tag{2.14}$$

These *univariate* statistics-based mappings, while simple to use, might overlook crucial details within the bags by not capturing dependencies between the features. If discriminative aspects do not significantly impact the overall statistics, they might be marginalized out in the aggregation, potentially leading to a loss of critical information for classification.

For example, the following two distinct bags $\mathcal{B}_1$ and $\mathcal{B}_2$ are indistinguishable after being embedded into a vector space using the above-defined mappings (Equation 2.14):

$$\boldsymbol{\Phi}(\mathcal{B}_1) = \boldsymbol{\Phi}\left(\begin{Bmatrix}(0,0)\\(1,1)\end{Bmatrix}\right) = (0,0,1,1), \qquad \boldsymbol{\Phi}(\mathcal{B}_2) = \boldsymbol{\Phi}\left(\begin{Bmatrix}(0,1)\\(1,0)\end{Bmatrix}\right) = (0,0,1,1). \tag{2.15}$$

Moreover, if one of the bags contained a unique, discriminative signal in the form of an instance $\mathbf{x} = (0.5, 0.5)$, that signal would not be extracted and consequently lost with these mappings.

More sophisticated vocabulary-based mappings, therefore, analyze how much a given bag $\mathcal{B}$ matches specific (possibly *multivariate*) patterns Φ_i, referred to as "words", from a pre-defined vocabulary $\{\Phi_1, \Phi_2, \ldots, \Phi_D\}$:

$$\boldsymbol{\Phi}(\mathcal{B}) = (\phi(\mathcal{B}; \Phi_1), \phi(\mathcal{B}; \Phi_2), \ldots, \phi(\mathcal{B}; \Phi_D)), \tag{2.16}$$

where $\phi(\mathcal{B}; \Phi_i) \to \mathbb{R}$ denotes a measure quantifying the matching between the bag $\mathcal{B}$ and the pattern Φ_i. This measure typically takes the form of $\phi(\mathcal{B}; \Phi_i) = \mathrm{agg}_{\mathbf{x}\in\mathcal{B}} \mathrm{k}(\mathbf{x}; \Phi_i)$, where agg stands for an aggregation function (e.g. minimum, mean or maximum), often called a pooling function, and $\mathrm{k}(\mathbf{x}; \Phi) \to \mathbb{R}$ is some sort of an instance-level "pattern-matching" measure, assessing the degree of correspondence or similarity between the instance $\mathbf{x}$ and the pattern Φ_i. Individual vocabulary-based methods then differ in the selection of the aggregation function, the pattern-matching measure, and the definition of vocabulary patterns.

Chen *et al.* [26] propose a MILES algorithm, where raw instances from positive training bags are used as the vocabulary patterns. Let us denote them as $\{\mathbf{x}_{\Phi 1}, \mathbf{x}_{\Phi 2}, \ldots, \mathbf{x}_{\Phi D}\}$. To measure the similarity between the bag $\mathcal{B}$ and a certain pattern $\mathbf{x}_{\Phi i}$ (i.e. a training instance), MILES uses the Gaussian basis function:

$$\phi(\mathcal{B}; \Phi_i) = \max_{\mathbf{x}\in\mathcal{B}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_{\Phi i}\|^2}{\sigma^2}\right), \tag{2.17}$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ is the squared Euclidean distance between two feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\sigma > 0$ is a scaling hyper-parameter. Observe that by identifying individual training instances within a given (testing) bag, we can effectively address the redefined keychain problem illustrated in Figure 2.2. Recall that in the redefined problem, a bag (keychain) must contain two specific types of instances (keys) to be labeled as positive. With this MILES approach, at least one pattern would be certainly associated with the green key, and one with the blue key, as both instances are present in the training set. This guarantees the extraction of the discriminative information needed for resolving the problem.

Sivic *et al.* [104] employ the K-Means algorithm to generate a vocabulary of prototype instances, rather than using all training instances. This helps to avoid scenarios where large training bags result in excessively large vocabularies crowded with many redundant/irrelevant patterns. The K-Means algorithm is used to group instances from all training bags into a predefined

number of D-distinct clusters, based on their similarity in the instance space. Centroids of these clusters, calculated as the mean of the instances in each cluster, then serve as the prototype instances: $\{\mathbf{x}_{\Phi 1}, \mathbf{x}_{\Phi 2}, \ldots, \mathbf{x}_{\Phi D}\}$. To create a single feature vector representation of bag $\mathcal{B}$, reflecting relations to individual vocabulary patterns Φ_i, the following computation is performed:

$$\phi(\mathcal{B}; \Phi_i) = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbb{1} \left[i = \underset{j=1,\ldots,D}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{x}_{\Phi j}\| \right], \tag{2.18}$$

where symbol $\mathbb{1}$ stands for an indicator function that equals one if its argument is true and zero otherwise. The argument of the minimum argmin ensures that each instance $\mathbf{x}$ is assigned to exactly one pattern Φ_j from the vocabulary based on the minimal Euclidean distance $\|\mathbf{x} - \mathbf{x}_{\Phi j}\|$ (i.e. hard assignment). This technique is known under the name of Histogram Bag-of-Words (H-BoW) because values of $\phi(\mathcal{B}; \Phi_i)$, where $i \in \{1, \ldots, D\}$, correspond to bins in a histogram $\mathbf{\Phi}(\mathcal{B})$ counting the number of instances in the bag that belong to each cluster (i.e. pattern). These counts are then optionally normalized by the bag size $|\mathcal{B}|$. H-BoW is a well-known technique with broad use in ML, particularly in the field of Computer Vision [83].

A popular variant of the H-BoW mapping technique is to generate the vocabulary of patterns using the probabilistic Gaussian Mixture Model (GMM) instead of the K-Means algorithm. In this approach, the Expectation-Maximization (EM) algorithm is applied to estimate the distribution of training instances as a sum of weighted Gaussian components: $\mathrm{P}(\mathbf{x}) = \sum_{i=1}^{D} w_i \, \mathcal{G}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $w_i \in \mathbb{R}$ is a weight, $\boldsymbol{\mu}_i \in \mathbb{R}^d$ a center, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{d \times d}$ a covariance matrix of $\mathcal{G}_i$-th Gaussian component. These components then form the vocabulary of patterns $\{\mathcal{G}_{\Phi 1}, \mathcal{G}_{\Phi 2}, \ldots, \mathcal{G}_{\Phi D}\}$. Since GMM is a probabilistic model, the partial mappings $\phi(\mathcal{B}; \Phi_i)$ of the final mapping function $\mathbf{\Phi}(\mathcal{B}) = (\phi(\mathcal{B}; \Phi_1), \ldots, \phi(\mathcal{B}; \Phi_D))$ are computed as follows:

$$\phi(\mathcal{B}; \Phi_i) = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathrm{P}(\mathcal{G}_{\Phi i}|\mathbf{x}), \quad \mathrm{P}(\mathcal{G}_{\Phi i}|\mathbf{x}) = \frac{w_{\Phi i} \, \mathcal{G}(\mathbf{x}|\boldsymbol{\mu}_{\Phi i}, \boldsymbol{\Sigma}_{\Phi i})}{\sum_{j=1}^{D} w_{\Phi j} \, \mathcal{G}(\mathbf{x}|\boldsymbol{\mu}_{\Phi j}, \boldsymbol{\Sigma}_{\Phi j})}, \tag{2.19}$$

where $\mathrm{P}(\mathcal{G}_{\Phi i}|\mathbf{x})$ is a posterior probability that instance $\mathbf{x}$ belongs to component $\mathcal{G}_{\Phi i}$ (i.e. soft-assignment). We successfully used this H-BoW-GMM technique in our prior work [60] for modeling encrypted HTTPS communications between network users and servers (bags), each consisting of multiple HTTPS requests (4-dimensional instances) made during these communications. It proved to be better in distinguishing between malicious and legitimate communications than a previously applied approach based on soft-histograms [121].

Pevný and Somol [91] propose MIL-NN, a neural network formalism for embedding and solving MIL problems. The architecture of the networks is as follows. One (or more) lower "instance" layers project instances $\mathbf{x}$ of bag $\mathcal{B}$ from the instance space $\mathbb{R}^d$ to the embedded bag space $\mathbb{R}^D$ where a pooling function (e.g. maximum) aggregates them into a single feature vector representation. This embedded bag representation is then projected to the output (classification) layer by higher "bag" layer(s) of the network as usual. Hence, each neuron in the introduced aggregation layer can be viewed as one vocabulary pattern:

$$\phi(\mathcal{B}; \Phi_i) = \max_{\mathbf{x} \in \mathcal{B}} \mathrm{ReLU}\left(\langle \mathbf{w}_{\Phi i}, \mathbf{x} \rangle + b_{\Phi i}\right), \tag{2.20}$$

where $\mathrm{ReLU}(x) = \max\{0, x\}$ is Rectified Linear Unit, an activation function of a particular choice, along with the maximum pooling function. But, other options for these functions can be used as well. Using the formula from Equation 2.13, all vocabulary patterns (neurons) constitute the mapping (aggregation) layer $\mathbf{\Phi}(\mathcal{B}) = (\phi(\mathcal{B}; \Phi_1), \ldots, \phi(\mathcal{B}; \Phi_D)) \in \mathbb{R}^D$, on top of which other layers implement the classifier $\mathrm{f}(\mathbf{\Phi}(\mathcal{B})) \to \{-1, +1\}$ that already uses the embedded single feature vector representation of the bag $\mathcal{B}$. The key advantage of this approach is that the back-propagation simultaneously optimizes the classifier and the embedding, effectively learning the optimal bag representation that suits the classification task. In Chapter 3, we will aim to develop a similar approach for ensembles of decision trees.

2.2 Hierarchical Multiple Instance Learning

Pevný, Somol, and Mandlík [90, 80] take the MIL formalism a step further by allowing individual data samples to be represented as hierarchical tree structures of variable breadth and depth. In addition to the standard MIL **bag** structure $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$, designed to hold a flexible number of order-independent, equal-dimension instances $\mathbf{x} \in \mathbb{R}^d$, they introduce a **product** structure $\mathcal{P}$ that can store a *fixed* number of order-*dependent* instances of potentially *varying* dimensions through vector concatenation: $\mathcal{P} = (\dot{\mathbf{x}}, \ldots, \ddot{\mathbf{x}}) \in \mathbb{R}^{\dot{d}+\ldots+\ddot{d}}, \dot{\mathbf{x}} \in \mathbb{R}^{\dot{d}}, \ldots, \ddot{\mathbf{x}} \in \mathbb{R}^{\ddot{d}}$. Moreover, individual instances $\mathbf{x}$ within both structures may represent either the atomic elements (strings, numbers, Boolean values, etc.) of the corresponding data sample (leaves of the tree) or (higher-level) embeddings of potential deeper sub-structures within the tree. This allows organizing the (lowest level) atomic instances into a tree structure by wrapping them into increasingly higher-level nested Bag/Product structures.

To process such structured data, a Hierarchical MIL Neural Network (HMIL-NN) model is built with an architecture that reflects the HMIL samples' tree topology. The idea is to create a hierarchy of embeddings — using feed-forward networks, permutation invariant aggregations, and vector concatenations — that gradually map atomic instances and subsequent higher-level Bag/Product structures to fixed-size vectors. Figure 2.3 depicts this process in a graphical form.

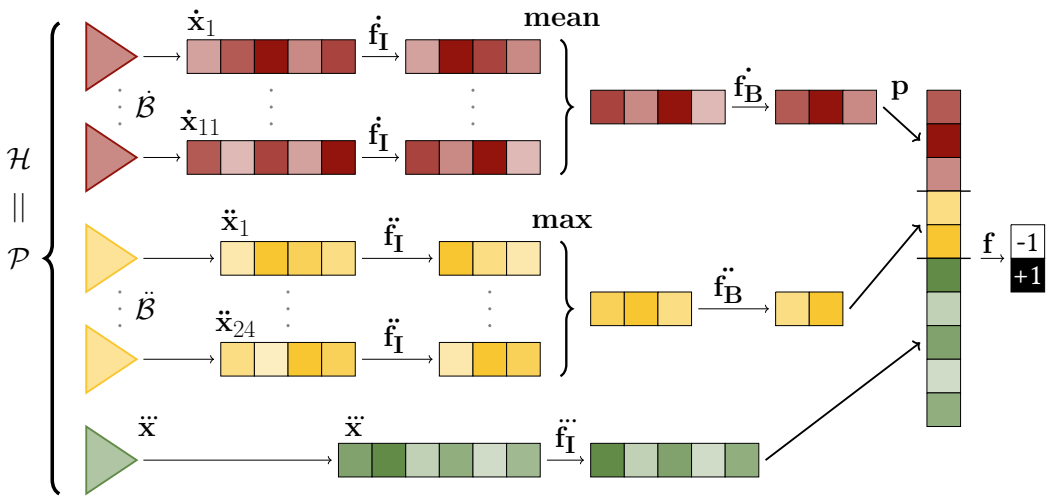

Figure 2.3: Classification of a **hierarchically structured data sample** $\mathcal{H}$ by a HMIL-NN model, where: $\mathcal{H} = (\{\dot{\mathbf{x}}_1, \ldots, \dot{\mathbf{x}}_{11}\}, \{\ddot{\mathbf{x}}_1, \ldots, \ddot{\mathbf{x}}_{24}\}, \dddot{\mathbf{x}}) = (\dot{\mathcal{B}}, \ddot{\mathcal{B}}, \dddot{\mathbf{x}})$. Instances, denoted as $\dot{\mathbf{x}}$, have the same dimensionality $\dot{\mathbf{x}} \in \mathbb{R}^5$, their order within bag $\dot{\mathcal{B}}$ does not influence the classification outcome, and their quantity may vary across samples in the dataset. The same applies to instances $\ddot{\mathbf{x}}$ within $\ddot{\mathcal{B}}$, except for their dimensionality, which is $\ddot{\mathbf{x}} \in \mathbb{R}^4$. The last type of instance, denoted as $\dddot{\mathbf{x}} \in \mathbb{R}^6$, appears only once in each sample within the dataset. All these types of instances may represent either atomic elements (e.g. string, numbers, etc.) or higher-level embeddings of potential deeper sub-structures. The entities $\dot{\mathcal{B}}$, $\ddot{\mathcal{B}}$, and $\dddot{\mathbf{x}}$ are further wrapped into a product $\mathcal{P}$, which respects element order and permits the embeddings of its elements to have varying dimensionalities. **The data flow through the HMIL-NN model is as follows**: First, instance-level feed-forward networks $\dot{\mathbf{f}}_{\mathbf{I}} : \mathbb{R}^5 \to \mathbb{R}^4$ and $\ddot{\mathbf{f}}_{\mathbf{I}} : \mathbb{R}^4 \to \mathbb{R}^3$ project instances of $\dot{\mathcal{B}}$ and $\ddot{\mathcal{B}}$ into new embedded spaces in which they are aggregated by mean and max pooling functions, respectively. Then, the final fixed-size vector representations of the bags are created by projecting the resulting vectors once more, with bag-level networks $\dot{\mathbf{f}}_{\mathbf{B}} : \mathbb{R}^4 \to \mathbb{R}^3$ and $\ddot{\mathbf{f}}_{\mathbf{B}} : \mathbb{R}^3 \to \mathbb{R}^2$. Finally, the concatenation function $\mathbf{p}$ combines the representations of the bags with the representation of the third instance type $\dddot{\mathbf{x}}$, which is obtained by feed-forward network $\dddot{\mathbf{f}}_{\mathbf{I}} : \mathbb{R}^6 \to \mathbb{R}^5$. This results in a final feature vector that represents the entire data sample $\mathcal{H}$. This vector is then fed into a classification network $\mathbf{f} : \mathbb{R}^{10} \to \mathbb{R}^2$ with two output class units.

2.3 Learning on JavaScript Object Notation documents

JavaScript Object Notation, or JSON for short, is a popular text-based data format for capturing structured information in a flexible tree-like form[1]. It is built upon six value types: four primitive data types, namely **string** (e.g. `"Hello World"`), **number** (e.g. `42`), **boolean** value (i.e. either `true` or `false`), and **null** (i.e. `null`); along with two collection types:

- **array**: A collection of *ordered* elements, each of which is one of the six JSON value types. The square brackets `"[]"` are used to symbolize JSON arrays, with individual elements separated by commas. For example, `["McKay", "Keller", "Carter"]`.
- **object**: A collection of *unordered* key-value pairs, where keys are strings and values are any of the six JSON value types. The curly braces `"{}"` are used to symbolize JSON objects, with each key-value pair separated by a colon and individual pairs distinguished by commas. For example, `{"name": "McKay", "age": 54}`.

Moreover, by nesting objects and arrays within each other, one can create more complex, hierarchical data structures, resembling a tree-like topology. In this tree topology, the leaves correspond to the primitive data types, while the higher-level nodes represent the objects and arrays that organize the data in leaves into a structured hierarchy.

```
{
  "name": "McKay",
  "age": 55,
  "career": "scientist",
  "retired": false,
  "friends" : [
    {
      "name": "Keller",
      "born": 1968
    },
    {
      "name": "Carter",
      "born": 1965
    }
  ]
}
```

Figure 2.4: An illustrative JSON data sample, capturing information about a person named McKay, as well as details regarding his friends, arranged in a 3-level JSON structure.

Figure 2.4 shows an example of a JSON data sample with a simple 3-level hierarchy. It is a JSON object that captures information about a person named McKay, together with details about his friends. The root of the tree hierarchy stands for the `"person"` object, which has five key-value pairs that represent different properties of the person. The key `"name"` holds the string `"McKay"`, `"age"` has the number `54`, `"career"` contains the string `"scientist"`, `"retired"` is a Boolean with the value `false`, and `"friends"` has an array of objects as its value. In the `"friends"` array, there are two objects, each representing a friend of McKay. These objects represent the second level of the hierarchy, while their corresponding values represent the third, and lowest, level within this structure. The first `"friend"` object contains `"name"` as `"Keller"` and `"born"` as `1968`, while the second has `"name"` as `"Carter"` and `"born"` as `1965`.

Although this is a simplistic example and real-world JSONs are typically much more complex and varied (refer to Chapter 6 for cybersecurity applications and Figures 6.4, 6.7, 6.9, and 6.11), it already illustrates the key advantages of the JSON format:

- JSON syntax is lightweight and designed for simple machine and human readability.
- JSON format is self-describing, meaning there is no need for separate schema definitions for used structures.
- JSON exhibits high mutability, allowing for in-place modifications and additions of key-value pairs directly within a particular JSON sample as required.
- JSON arrays offer size flexibility, capable of handling an arbitrary number of elements without an upper limit.

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

- JSON has inherent support for missing values, meaning that not all key-value pairs need to be specified in every JSON sample.
- JSON, thanks to its versatile structure and variety of supported data types, can represent a broad spectrum of complex objects, scenarios, and relationships.

While these characteristics make JSON an excellent format for data representation and manipulation, they also contribute to its non-vectorial nature, which complicates its use in ML applications. To the best of our knowledge, there are only two existing methods that facilitate automated model learning directly on JSON data, thus eliminating the need for expensive and often lossy manual feature engineering.

Woof and Chen [113] present STRLA, a Semantic Tree-structured Recursive Learning Architecture for end-to-end learning on generic semantic tree-structured data of arbitrary topologies and heterogeneous data types, such as data expressed in JSON. Inspired by the concept of a recursive neural network, STRLA assigns each node in a tree a latent hidden-state vector. This vector depends on the data linked to the current node and the hidden states of its children nodes, if any. The same neural network component is recursively applied to each node of the tree, starting from the leaves and moving upwards to the root. This ensures that every node in the tree eventually gets assigned a hidden state. To manage data heterogeneity in JSON leaves, the framework incorporates a range of embedding functions to handle different primitive types separately. These functions are mapping the four JSON primitive data types (i.e. number, string, Boolean, and null) to a d-dimensional latent representation. As for JSON objects and arrays, the framework employs another set of functions that can manage input from an arbitrary number of latent vectors. There are two main implementations of the STRLA algorithm, resulting from different types of these functions: implementation based on Deep-set networks [116] and Long Short-Term Memory (LSTM)-based implementation [56], here and after denoted as JSON-NN. The ability to take into account the order of elements in JSON arrays is the main advantage of JSON-NN over the deep-set-based approach.

Mandlík *et al.* [122] introduce JsonGrinder, an automated feature extraction technique for JSONs that leverages the Hierarchical MIL paradigm (Section 2.2) and the capabilities of HMIL-NN models [80]. Given a dataset of JSON samples, JsonGrinder first estimates the JSON schema shared by the samples. Based on this schema, it transforms each JSON sample into an HMIL data sample — a nested composition of bag and product structures encapsulating atomic instances that represent the underlying values stored in JSON leaves. The conversion process translates JSON arrays into bag structures, and JSON objects into product structures. Values in JSON leaves are transformed into instances through various methods, utilizing either custom extractors or suggested extractors by JsonGrinder. These are based on heuristics derived from the statistical profile of the JSON leaf values in the sample corpus. For example, diverse collections of strings are represented as n-gram histograms and small collections of unique string values as one-hot encoded categorical variables; numbers are maintained in their original form. Potentially missing JSON values (i.e. individual leaves or even entire JSON sub-structures) relative to the deduced JSON schema are managed in JsonGrinder by explicitly storing them as missing values. Then, during training and prediction, these missing values are replaced with unique, trainable imputations unique to each Bag/Product structure. Finally, JsonGrinder suggests an architecture for the HMIL-NN model, capable of processing the generated HMIL samples. Figure 2.5 depicts a simplified visualization of the JSON sample from Figure 2.4 being processed by JsonGrinder and HMIL-NN.

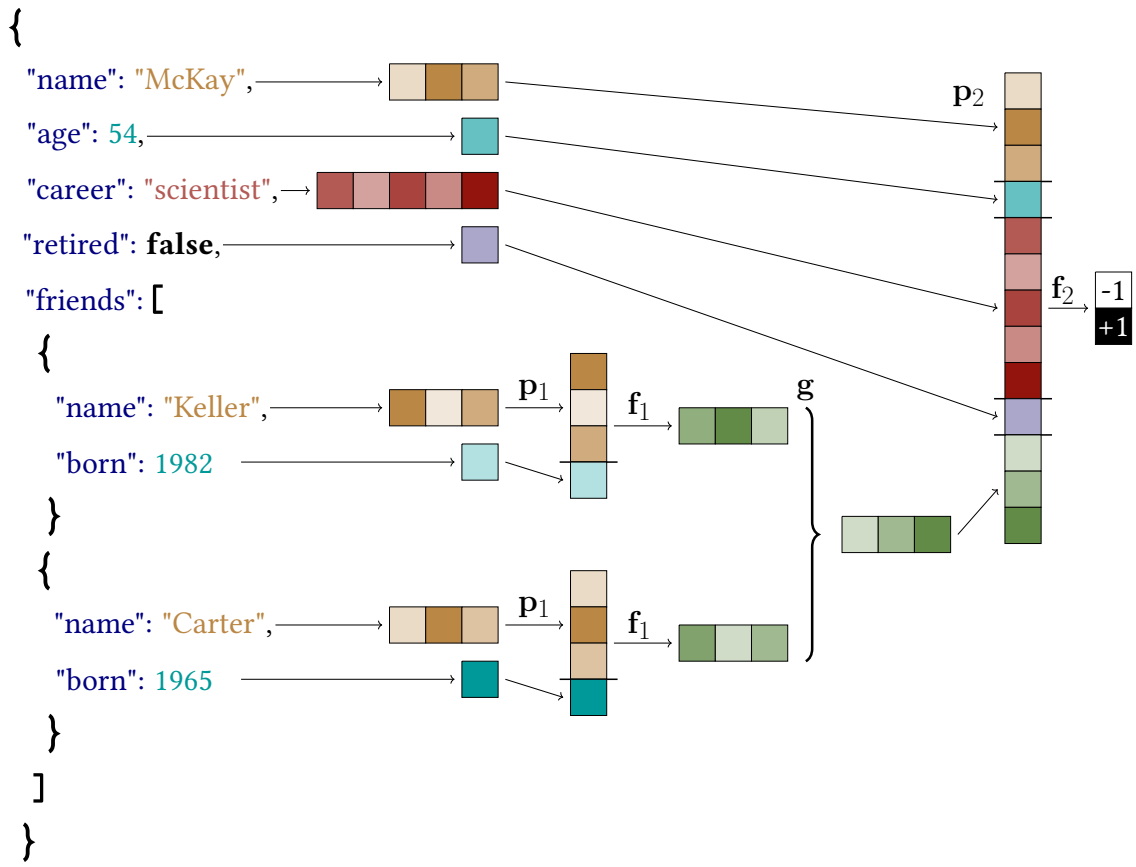

Figure 2.5: A diagram illustrating the process of classifying a JSON sample (seen in Figure 2.4) using JsonGrinder and HMIL-NN. **Initially, JsonGrinder transforms** the JSON sample into an HMIL structure that is compatible with HMIL-NN. In doing so, each leaf in the JSON tree (containing a primitive data type) is converted into a numerical instance. This conversion uses suitable feature extraction techniques, such as n-grams for strings (i.e. the "name" fields), direct value representation for numbers and the Boolean value (i.e. the "age", "born" and "retired" fields), and one-hot encoding for the categorical value (i.e. "scientist"). These atomic instances are then encapsulated into respective higher-level bag/product HMIL structures (see Section 2.2) according to the type of collection they belong to — bag structure for JSON arrays and product structure for JSON objects. The lowest-level instances of the two "friend" objects (specifically, the "name" and "born" fields) are first wrapped into the product structure, which corresponds to the JSON "friend" object. These are then packaged into the bag structure, which aligns with the JSON "friends" array. Ultimately, in conjunction with the remaining highest-level instances, they are all integrated into the product structure (analogous to the root "person" object) that represents the entire JSON sample. **Then, HMIL-NN processes** the newly created HMIL representation of the JSON sample. The architecture of this classifier is specifically designed (by JsonGridner) to mirror the schema of the JSON sample and other samples from a respective dataset. First, the instances within the "friend" product structures are concatenated with $\mathbf{p}_1$ and projected into a new embedded space via the first feed-forward network $\mathbf{f}_1 : \mathbb{R}^4 \rightarrow \mathbb{R}^3$ (which is shared for all "friend" objects). These instances are then aggregated using a pooling function $\mathbf{g}$ (e.g. mean or maximum) to form a single feature vector representation of the top-level "friends" field. Finally, all instances representing the top-level fields of the "person" object (being classified) are concatenated by $\mathbf{p}_2$ and sent to the second feed-forward network $\mathbf{f}_2 : \mathbb{R}^{13} \rightarrow \mathbb{R}^2$ performing the class label prediction.

Chapter 3

Bag-level Randomized Trees

In this chapter, we present our first tree-based algorithm for solving MIL problems, where individual data samples $\mathcal{B}_1, \mathcal{B}_2, \ldots$ (called bags) are represented by groups of multiple d-dimensional feature vectors $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\},\ \mathbf{x} \in \mathbb{R}^d$ (called instances), instead of single feature vectors.

Although many MIL algorithms have been already developed, most of them work well only in their specific application domains. As demonstrated in the recent study of MIL problems [28] (and Table 3.2 shows it too), existing MIL classifiers do not perform as well over a broader range of problems and/or require very careful hyper-parameter tuning. Ensembles of decision trees like Breiman's Random Forests [18] or Extremely Randomized Trees [47] are, on the other hand, known to be reasonably effective over many domains even with their default hyper-parameter settings [42], which makes them a good case for MIL adaptation. However, the prior-art adaptations of decision trees to the MIL setting like MITI [15], MIForest [75] and MIOForest [108], with the exception of RELIC [100], are representatives of the early *instance-level* era. Methods from this period of research are generally outperformed by their later *bag-level* counterparts working under the more realistic assumption that the discriminative information can be scattered across more than one instance in a bag [5]. This fact inspired us to propose a new MIL algorithm based on the *ensemble of decision trees* that would operate on the *bag-level* and so combine the best of both worlds.

3.1 Algorithm Description

The Bag-level Randomized Trees (BLRT) are trained according to the classical top-down greedy procedure for building ensembles of unpruned decision trees. Individual tree learners recursively partition a training dataset by choosing binary splitting rules until pure sample sets are obtained (Figure 3.1).

The key difference, however, lies in the conditions that are evaluated inside the splitting nodes. While nodes of standard single-instance decision trees $\mathcal{N}_{\text{SIL}}$ (Equation 3.1) evaluate only whether feature f of a given sample $\mathbf{x}$ is greater than certain value v, nodes of the proposed bag-level randomized trees $\mathcal{N}_{\text{BLRT}}$ also count the number of instances within the sample (i.e. bag) that accomplish the condition. This absolute count is then normalized by bag size $|\mathcal{B}|$ and compared to value $r \in [0, 1)$ (Equation 3.2).

$$\mathcal{N}_{\text{SIL}}(\mathbf{x}; f, v) = \begin{cases} \text{left}, & \text{if } x_f > v, \\ \text{right}, & \text{otherwise.} \end{cases} \tag{3.1}$$

$$\mathcal{N}_{\text{BLRT}}(\mathcal{B}; \underbrace{f, v, r}_{\Phi}) = \begin{cases} \text{left}, & \text{if } \left[\frac{1}{|\mathcal{B}|}\sum_{\mathbf{x}\in\mathcal{B}} \mathbb{1}\left[x_f > v\right]\right] > r, \\ \text{right}, & \text{otherwise.} \end{cases} \tag{3.2}$$

Parameter r denotes a relative count of instances $\mathbf{x}$ inside bag $\mathcal{B}$ that must satisfy the inner condition $x_f > v$ to be the whole bag passed to the left branch. It is the only additional parameter that needs to be learned from the training data together with f and v. Symbol $\mathbb{1}$ stands for an indicator function that equals one if its argument is true and zero otherwise.

Note that if bags are of size one, nodes $\mathcal{N}_{\text{BLRT}}$ behave like the traditional $\mathcal{N}_{\text{SIL}}$ regardless of the value of r parameters. The next special case is when the relative count takes extreme values, i.e. $r \in \{0, 0.\bar{9}\}$[1]. The proposed algorithm then becomes equivalent to the prior art solution known as RELIC [100]. Under this condition, the splitting rules act as either the universal or the existential quantifier. In particular, bags are tested in two possible ways: if there exists at least one instance that fulfills the inner condition or if the condition is satisfied by all instances. An experiment in Section 3.2 (Figure 3.3), however, shows that the ability of the proposed algorithm to test situations also between these two extreme cases is highly beneficial on many datasets.

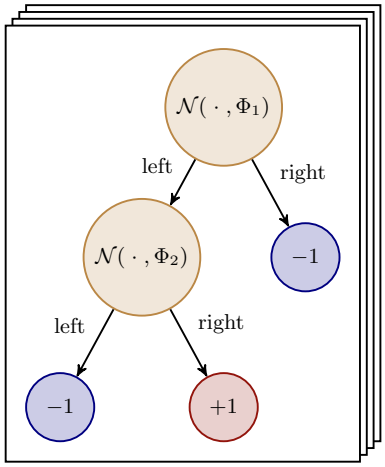

Figure 3.1: Ensemble of decision trees with a learned structure and splitting parameters $\{\Phi_1, \Phi_2, \dots\}$.

The search for optimal splitting parameters $\Phi^* = (f, v, r)$ during the tree growth is implemented in a randomized manner. At each node construction, a set of candidate splitting rules $\mathcal{R} = \{\Phi_1, \dots\}$ is generated (based on local training subset $\mathcal{S} \subseteq \mathcal{D}$) among which the best one Φ^* is selected according to a score obtained by an impurity measure such as Information gain [94] or Gini impurity [19]. Specifically, for each feature f out of K randomly selected ones, T values of parameter v are drawn uniformly from interval $[x_f^{\min}, x_f^{\max})$, where $x_f^{\min}$ and $x_f^{\max}$ denote the minimum and the maximum value of feature f across all bags within the local sample set $\mathcal{S}$. For each such pair (f, v), other T values of parameter r are generated uniformly from interval $[0, 1)$[2]. In total, there are $K \times T \times T$ candidate splitting rules at maximum[3]. A detailed description of the tree induction procedure is outlined in Algorithm 1 in the form of pseudo-code.

The above randomized approach is adopted from Extremely[4] Randomized Trees [47] and generalized to MIL setting by adding the third parameter r (i.e. the relative count). Unlike CART algorithm, used e.g. in Breiman's Random Forests [18], the randomized search does not require going through all possible splitting points on selected features, which could be prohibitively expensive in this MIL variant of trees. Furthermore, the explicit randomization in combination with ensemble averaging makes the decision boundary more smooth, resulting in models with better or equal accuracy than that of Random Forests [47].

[1] Technically, the value of $0.\bar{9}$ should be one minus the smallest representable value.

[2] Interestingly, a slight but consistent improvement in performance can be obtained by independent generation of v and r values. Similarly, when using stochastic hyper-parameter search as opposed to grid one, more distinct values of each parameter are explored given the same amount of trials [14].

[3] If $x_f^{\min}$ equals to $x_f^{\max}$, no splitting rules are generated on feature f.

[4] Term *extremely* corresponds to setting $T = 1$.

Algorithm 1: Induction algorithm for Bag-level Randomized Trees (binary classification problem $y \in \{-1, +1\}$ and numerical features are assumed).

Function Train($\mathcal{D}; M, K, T$)

Input : Training set $\mathcal{D} = \{(\mathcal{B}, y)_1, \ldots\}$,
number of trees to grow M,
number of randomly selected features K,
number of generated thresholds T.
Output: Ensemble of Bag-level Randomized Trees $\mathcal{E}$.

$\mathcal{E} = \emptyset$
foreach tree in $1 \ldots M$ **do**
  $\mathcal{E} = \mathcal{E} \cup \{\text{BuildTree}(\mathcal{D}; K, T)\}$
return $\mathcal{E}$

Function BuildTree($\mathcal{S}; K, T$)

Input : Local training subset $\mathcal{S} \subseteq \mathcal{D}$.
Output: A node with left and right followers or a leaf.

if all y in $\mathcal{S}$ are equal **then**
  return leaf(y)
$\mathcal{R} = \text{GenerateCandidateSplittingRules}(\mathcal{S}; K, T)$
$\Phi^* = \text{argmax}_{\Phi \in \mathcal{R}}\, Score(\mathcal{S}, \Phi)$
$\mathcal{S}_{\text{left}} = \{\mathcal{B} \in \mathcal{S} | \mathcal{N}_{\text{BLRT}}(\mathcal{B}; \Phi^*) = \text{left}\}$
$\mathcal{S}_{\text{right}} = \mathcal{S} \setminus \mathcal{S}_{\text{left}}$
if $\mathcal{S}_{left} = \emptyset$ *or* $\mathcal{S}_{right} = \emptyset$ **then**
  return leaf($\frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} y$)
return node($\mathcal{N}_{\text{BLRT}}(\,\cdot\,; \Phi^*)$, BuildTree($\mathcal{S}_{\text{left}}$), BuildTree($\mathcal{S}_{\text{right}}$))

Function GenerateCandidateSplittingRules($\mathcal{S}; K, T$)

Output: Set of candidate splitting rules $\mathcal{R} = \{\Phi_1, \ldots\}$.

$\mathcal{R} = \emptyset$
foreach feature f in K randomly selected ones (without replacement) **do**
  Find extremes $x_f^{\min}$ and $x_f^{\max}$ on given feature f across all bags $\mathcal{B} \in \mathcal{S}$
  if $x_f^{min} \neq x_f^{max}$ **then**
    foreach value v in T uniformly drawn values from $[x_f^{\min}, x_f^{\max})$ **do**
      foreach value r in T uniformly drawn values from $[0, 1)$ **do**
        $\mathcal{R} = \mathcal{R} \cup \{\Phi\}$, $\Phi = (f, v, r)$
return $\mathcal{R}$

Algorithm 1 builds M fully grown decision trees. Each tree is trained on the whole sample set rather than a bootstrap replica[5] as realized e.g. in Random Forests. The reason is that training on the full sample set minimizes bias. Variance is reduced by the *strong* randomization in the splitting parameters combined with the output aggregation across multiple trees. Additional randomization in the form of *bagging* (with the aim to further increase independence among trees and thus reduce the variance of the ensemble) is therefore unnecessary for Randomized Trees and usually only deteriorates performance due to the increased bias [47][6].

From the computational point of view, the time complexity of the learning procedure is, assuming balanced trees, $\Theta(MKT^2N_I \log N_\mathcal{B})$, where $N_\mathcal{B}$ and N_I denote the number of bags and the number of instances within the bags, respectively. When bags are of size one (i.e. $N_I = N_\mathcal{B} = N$) and $T = 1$, the complexity is equivalent to the complexity of Extremely Randomized Trees $\Theta(MKN \log N)$ [47].

In the testing mode, assuming a binary classification problem (i.e. $y \in \{-1, +1\}$), predictions of individual trees are aggregated by a simple arithmetic average to produce a final prediction score $\hat{y} \in [-1, 1]$.

3.2 Evaluation on Public Datasets

The proposed BLRT algorithm is evaluated on 29 real-life datasets that are publicly available, for instance, at https://doi.org/10.6084/m9.figshare.6633983.v1. The datasets with meta descriptions are listed in Table 3.1. These classification problems are well known and cover a wide range of conditions in terms of application domains (molecule, scene, image, text, audio spectrogram, etc.), ratios of positive and negative samples (e.g. Corel datasets with 0.05 imbalance ratio), feature dimensions (from 9 to 6519) and average numbers of bag instances (from 4 to 185). More details about the datasets can be found in the recent study of MIL datasets [28].

The same collection of datasets was also used in the evaluation of 28 MIL classifiers (including their variants) implemented in the MIL toolbox [109]. The last two columns of Table 3.1 summarize the results from the evaluation available also through http://homepage.tudelft.nl/n9d04/milweb/. We report only those classifiers that achieved the highest performance by means of AUC metric[7] at least on one problem. This selection results in 13 classifiers that are listed in Table 3.2 together with references to their original papers.

Since an exact experimental protocol is provided as a part of the referenced evaluation, we followed that protocol precisely. For each dataset, the protocol provides indexes of all splits in 5-times repeated 10-fold cross-validation. The material, however, does not specify any approach for hyper-parameter optimization. Therefore, we evaluated the proposed model using default parameter settings. We set the number of trees to grow to $M = 500$ which should ensure convergence of the ensemble, the number of randomly selected features at each split to the square root of the feature dimension $K = \sqrt{D}$, which is the default value for tree-based models, and the number of uniformly drawn values of v and r to $T = 8$.

Table 3.1 summarizes results from the evaluation in terms of average scores and standard deviations. Although among the prior art (28 MIL classifiers) there is no single winning solution and almost every problem is associated with a different classifier, which demonstrates the

[5] Tree bagging [18] is a training procedure where each tree is grown from a bootstrap replica obtained by random sampling with replacement in the original training sample set. Perturbations in the training set introduce a randomization effect reducing the variance of ensemble models. This variance reduction, however, comes at expense of increased bias.

[6] When working on [62], we observed that tree bagging can still be useful in scenarios with noisy labels. Training on bootstrap replicas ensures that not every tree is affected by all mislabeled samples. Bagging is therefore an optional parameter in our BLRT implementation.

[7] Area Under a ROC Curve showing the true positive rate as a function of the false positive rate. AUC is agnostic to class imbalance and classifier's threshold setting.

Dataset				BLRT		Best prior-art	
Name	Bags +/-	Feat.	Avg.inst.	AUC		AUC	Algorithm
Musk1	47/45	166	5	**96.8** (1.6)	*	92.9 (1.3)	MI-SVM g
Musk2	39/63	166	65	91.2 (1.8)	*	**95.3** (1.5)	MILES g
C. African	100/1900	9	4	**96.2** (0.2)		95.7 (0.4)	minmin
C. Beach	100/1900	9	4	**98.9** (0.2)	*	90.7 (0.9)	RELIC
C. Historical	100/1900	9	4	**99.2** (0.1)	*	92.9 (0.5)	EM-DD
C. Buses	100/1900	9	4	97.4 (0.2)	*	**99.5** (0.1)	minmin
C. Dinosaurs	100/1900	9	4	96.4 (0.1)	*	**99.9** (0.0)	MILES p
C. Elephants	100/1900	9	4	**97.1** (0.1)		96.9 (0.2)	minmin
C. Food	100/1900	9	4	**99.4** (0.1)	*	97.2 (0.2)	minmin
Fox	100/100	230	7	**73.3** (1.4)	*	69.8 (1.7)	MILES g
Tiger	100/100	230	6	**92.6** (1.0)	*	87.2 (1.7)	MILES g
Elephant	100/100	230	7	**95.8** (0.9)	*	91.1 (1.2)	MI-SVM g
Protein	25/168	9	138	74.9 (2.3)	*	**89.5** (1.4)	minmin
Harddrive1	191/178	61	185	**99.6** (0.2)	*	98.6 (0.1)	MILES g
Harddrive2	178/191	61	185	**99.5** (0.1)	*	98.6 (0.2)	RELIC
Mutagenesis1	125/63	7	56	**92.1** (1.3)		91.0 (0.5)	cov-coef
Mutagenesis2	13/29	7	51	**86.0** (3.5)		84.0 (3.4)	EMD
B. BrownCreeper	197/351	38	19	**99.5** (0.0)	*	96.5 (0.5)	RELIC
B. WinterWren	109/439	38	19	**99.8** (0.1)	*	99.3 (0.1)	summin
B. Pacifics.	165/383	38	19	**96.1** (0.2)	*	95.7 (0.3)	MILES g
B. Red-breasted.	82/466	38	19	**99.2** (0.2)		98.7 (0.4)	MILBoost
UCSBBreast.	26/32	708	35	84.5 (2.5)	*	**92.2** (3.1)	cov-coef
Newsgroups1	50/50	200	54	78.8 (2.6)	*	**89.8** (1.6)	meanmin
Newsgroups2	50/50	200	31	63.0 (4.0)	*	**78.1** (1.4)	meanmean
Newsgroups3	50/50	200	52	76.3 (4.1)		**77.4** (1.5)	meanmean
Web1	21/54	5863	29	86.5 (2.6)	*	**91.9** (0.0)	MI-SVM g
Web2	18/57	6519	30	50.7 (7.8)	*	**90.1** (0.5)	MI-SVM g
Web3	14/61	6306	34	73.4 (6.7)	*	**91.8** (0.4)	MI-SVM g
Web4	55/20	6059	31	80.0 (4.2)	*	**99.4** (0.0)	mean-inst

Table 3.1: Metadata about 29 used datasets together with classification scores and standard deviations presented in percent (AUC $\times$ 100). The best results are in boldface. Stars denote statistically significant ($\alpha = 0.05$) differences according to Welch's t-test.

Rank position														Avg.		
1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	rank	N/A	Algorithm
17	1	0	3	2	2	1	1	0	0	2	0	0	0	3.1	0	BLRT [ours]
0	2	1	4	2	1	0	5	3	0	3	0	1	2	7.5	5	EM-DD [118]
0	1	1	6	4	1	1	1	3	2	2	3	1	1	7.5	2	MILBoost [110]
3	3	3	5	1	1	0	2	2	1	0	3	1	1	6.0	3	MI-SVM g [7]
1	0	2	2	5	2	6	6	1	2	0	1	0	0	6.5	1	MILES p [26]
2	5	6	1	0	2	1	1	2	0	1	3	0	4	6.5	1	MILES g [26]
1	5	3	1	2	2	2	1	4	2	2	1	1	2	6.9	0	mean-inst [49]
1	2	1	2	0	0	8	4	1	5	0	3	2	0	7.8	0	cov-coef [49]
0	3	2	0	0	3	0	1	3	6	3	4	4	0	8.9	0	RELIC [100]
2	3	2	1	3	4	4	0	0	2	5	0	1	1	6.7	1	minmin [27]
0	2	2	2	0	2	3	1	2	2	5	4	2	1	8.6	1	summin [27]
0	2	3	4	5	3	0	1	1	2	2	2	1	1	6.7	2	meanmin [27]
3	0	0	1	2	1	0	3	3	4	1	3	5	1	8.9	2	meanmean [27]
0	1	2	1	2	1	2	0	2	0	2	1	2	0	7.5	13	EMD [117]

Table 3.2: Number of times that each algorithm obtained each rank in the evaluation on 29 benchmark datasets. For clarity, the table shows only a subset of 13 out of 29 prior-art classifiers that qualified for the evaluation. Letters p/g next to MI-SVM and MILES algorithms denote the used polynomial/Gaussian kernel, respectively. Column N/A indicates the number of absent evaluations for a particular method. Our BLRT has the best average rank of 3.1. The second lowest rank is nearly twice as high (6.0) and belongs to MI-SVM with a Gaussian kernel.

difficulty and diversity of MIL problems, the proposed model was able to outperform the best prior art algorithm for a given dataset in 17 out of 29 cases. The most significant improvement with respect to the prior art is on the group of image classification problems (Fox, Tiger and Elephant) and on some scene classification problems (Corel Beach and Corel Historical). On the other hand, the proposal is less accurate on text classification problems (Newsgroups[8] and Web), Protein and Breast datasets.

From Table 3.2 showing the ranking of algorithms in the evaluation, it can be observed that the second best classifier with the lowest average rank (MI-SVM [7] with Gaussian kernel) ranked first only three times. Overall, the proposed algorithm works very reliably even without any hyper-paramter tuning. Indeed, the proposal never ended on any of the last three positions, which is unique among all classifiers. It should be stressed though that not all prior art classifiers were evaluated on all 29 datasets. Column N/A of Table 3.2 indicates the number of missing evaluations.

The non-parametric Wilcoxon signed ranks test [32] (testing whether two classifiers have equal performance) confirmed at significance level $\alpha = 0.05$ that the proposed bag-level randomized trees are superior to any other involved method. The test compared pair-wisely the proposal with every prior art method, each time using an intersection of their available datasets. The two most similarly performing methods are mean-inst [49] (p-value 0.037) and MI-SVM [7] with Gaussian kernel (p-value 0.022).

Besides the above evaluation, we also provide a comparison to other tree-based MIL algorithms in Table 3.3, namely RELIC [100], MIOForest [108], MIForest [75], MITI [15] and RF [18]. Except for RELIC, all of them operate on *instance-level*; labels are assigned to instances and a bag is positive if it contains at least one positive instance. RF represents a naive approach where

[8] Except for Newsgroup3 where the proposal is competitive with the best prior art.

standard single-instance Random Forests are trained directly on instances that inherited bag labels. Reported classification accuracies in Table 3.3 are taken from the work of MIOForest [108]. Unfortunately, the classifiers were evaluated only on five pioneering datasets (i.e. Musk1-2 and the image classification problems) and their implementations are not publicly available. Nevertheless, as can be seen from Table 3.3, the proposal outperforms all the prior tree-based MIL solutions on these datasets.

Dataset	BLRT [ours]	RELIC	MIOForest	MIForest	MITI	RF
Musk1	**96**	83	89	85	84	85
Musk2	**91**	81	87	82	88	78
Fox	**75**	66	68	64	N/A	60
Tiger	**90**	78	83	82	N/A	77
Elephant	**93**	80	86	84	N/A	74

Table 3.3: Comparison with other tree-based MIL classifiers. Scores refer to accuracy in percent (ACC $\times$ 100). The prior art results are taken from the work of MIOForest [108].

3.3 Ablation Study

In Figure 3.2, we assess various variants of the proposed algorithm. Dots in each subplot represent the 29 datasets. Their (x, y) coordinates are given by AUC scores obtained by the tested variants. If a dot lies on the diagonal (i.e. $x = y$ line), there is no difference between the two tested variants from that particular dataset perspective. The first two **subplots (a-b)** illustrate the influence of the ensemble size. It can be observed that it is significantly better to use 100 trees than 10 trees, but building 500 trees usually does not bring any additional performance. Also, according to **subplot (c)**, there is almost no difference between Information gain [94] and Gini impurity measure [19] scoring functions for selecting splitting rules. The next **subplot (d)** indicates that using higher values (e.g. 16 instead of the default 8) for parameter T (i.e. the number of randomly generated values for parameters v and r at each split) might lead to over-fitting on some datasets. In **subplot (e)** we tested a variant with an absolute count[9] instead of the relative one used in Equation 3.2. However, the variant with the absolute count performed significantly worse on most datasets. The last **subplot (f)** compares the proposed algorithm with its simplified alternative, where traditional Random Forests are trained on a non-optimized bag representation. To do so, all bags $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$ are transformed into single feature vectors $\{\mathbf{b}_1, \mathbf{b}_2, \ldots\}$ of values $b_{\mathcal{B}}^{(f,v)} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbb{1}\left[x_f > v\right]$, where for each feature f eight equally-spaced values v are generated from interval $[x_f^{\min}, x_f^{\max})$ that is estimated beforehand on the whole training sample set. As a result, the non-optimized bag representation is eight times longer than the dimensionality of instances. As seen from subplot (f), the Random Forests trained on the non-optimized bag representation are far inferior to the proposed algorithm on all datasets except one. This result highlights the importance to simultaneously optimize the representation parameters with the classification ones as proposed in Section 3.1.

Finally, Figure 3.3 shows histograms of learned values of r parameters for some datasets. The first observation is that datasets from the same source (e.g. Fox, Tiger and Elephant) have very similar distributions. This demonstrates that the learned knowledge of randomized trees is not as random as it might appear from the algorithm description. The next observation is that in almost all histograms (except for Mutagenesis problems), one or both extreme values of the parameter (i.e. $r \in \{0, 0.\bar{9}\}$) are the most frequent ones. As discussed in Section 3.1, the

[9] The sum in Equation 3.2 is not normalized by bag size $|\mathcal{B}|$ and parameter r can take values from interval $[1, \operatorname{argmax}_{\mathcal{B} \in \mathcal{S}} |\mathcal{B}|)$.

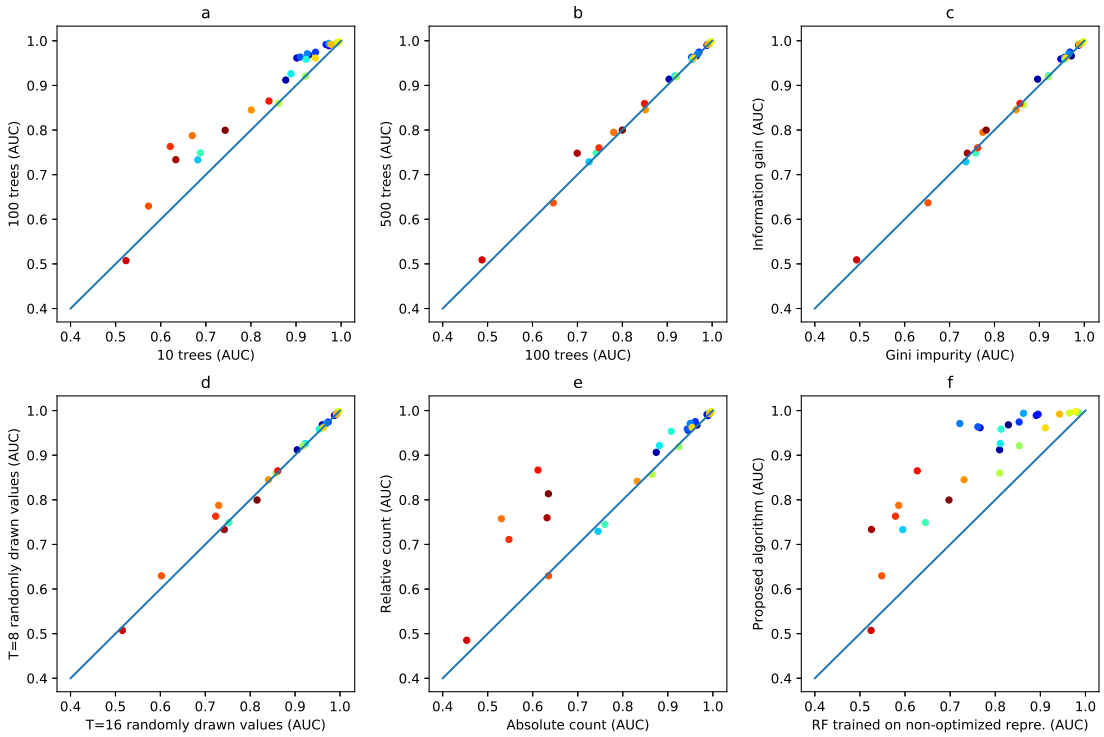

Figure 3.2: Pair-wise comparisons of various configurations of the proposed algorithm on the 29 datasets. Subplots **(a-b)** illustrate the influence of the ensemble size, subplot **(c)** the impact of selected impurity measure, subplot **(d)** the effect of parameter T, subplot **(e)** the performance of the variant with the absolute count and subplot **(f)** compares the proposed algorithm with RF trained on the non-optimized bag representation.

behavior of splitting rules (Equation 3.2) with extreme values is approaching the behavior of the universal or existential quantifier. On Web and Newsgroup datasets, this behavior is even dominant, meaning that the algorithm reduces to the prior art solution RELIC [100]. In the rest cases, however, the added parameter enabled learning of important dataset properties, which is supported by the high-level performance of BLRT.

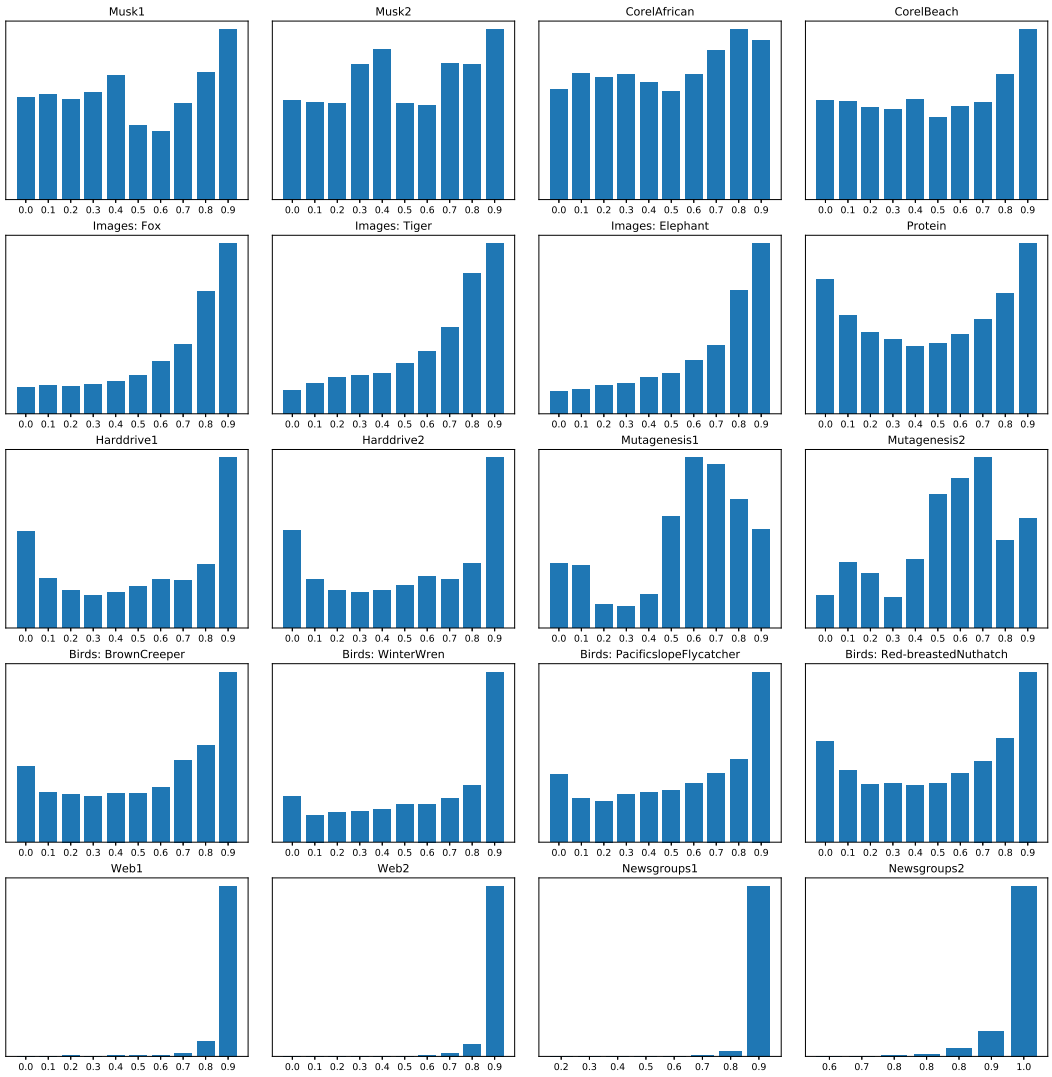

Figure 3.3: Histograms of learned values of r parameter (Equation 3.2) as collected across all tree-nodes in trained BLRT on a dataset. Datasets from the same source (e.g. Musk1-2, Harddrive1-2 and so forth) typically have very similar distributions that differ from the others. Although utilizing a randomized optimization in BLRT, this demonstrates consistency in learning dataset features. The dominance of extreme values on Web and Newsgroups means that BLRT behaves like RELIC [100] on these datasets.

3.4 Summarizing Comments

We have proposed a new tree-based algorithm for solving MIL problems called Bag-level Randomized Trees (BLRT). The algorithm naturally extends traditional single-instance decision trees, since bags with single instances are processed in the regular single-instance tree way. Multiple instance bags are judged by counting the percent of their instances that accomplish the standard condition testing whether a feature value is greater than a certain threshold. Judging this percent value is done through an additional parameter learned during the tree-building process.

Extreme values of the parameter reduce BLRT to the prior art solution RELIC [100]. Unlike other prior art tree-based algorithms, BLRT operates on the *bag-level*. The ability to analyze global bag-level information is most likely responsible for the superior performance. On the other hand, the algorithm is unable to identify instances that are responsible for positive bag

predictions, which may be required in some applications such as detecting malware-infected users in computer networks [64] or object tracking [10].

The algorithm falls into the category of *embedded-space* methods, since the learning procedure can be decoupled into two steps: embedding bags into single feature vectors and training traditional decision trees on top of the new representation. Features of the new single-vector representation then correspond to the counted percent values, which may alternatively be interpreted as quantile-based embedding. To obtain a meaning-full representation, a large set of candidate features can be firstly randomly generated and subsequently reduced to reasonable numbers using a fast feature selection method (e.g. CMIM [43]) as we did in our previous works [69, 68, 66]. However, the herein-presented approach optimizes both the representation and the tree classifier in a single step.

As a side effect, the algorithm inherits all desirable properties of tree-based learners. It is assumption-free, scale-invariant and robust to noisy and missing features. It can handle both numerical and categorical features. And, it can be easily extended to multi-class and regression problems.

Chapter 4

Instance Selection Randomized Trees

In this chapter, we introduce our second tree-based algorithm for solving multiple-instance classification problems. This time with an emphasis on explainability.

Since MIL formalism only requires labels for bags and not for the more numerous instances, it is especially useful in fields with high label acquisition costs. However, in domains where inference transparency is demanded, such as in cybersecurity, the sample attribution requirements are often asymmetric with respect to the training/application phase. While in the training phase it is very convenient to supply labels only for bags, in the application phase it is generally not enough to just provide decisions on the bag-level because the inferred verdicts need to be explained on the level of individual instances.

Unfortunately, the majority of MIL classifiers do not focus on this real-world need. On one end of the spectrum, there are *instance-level* methods (e.g. MI-SVM [7]) making decisions based on single instances, which is good for comprehension but often detrimental to prediction accuracy. And on the other end, there are *bag-level* methods (e.g. BLRT or MIL-NN [91]) with high accuracy but usually bad interpretability due to the usage of some aggregation blending information from all instances together.

We, therefore, propose a novel method that is forced to select only instances that are important for making decisions, leaving the rest untouched. Depending on the ratio of selected and untouched instances, the model oscillates between the instance-level and bag-level approaches. The higher the number of untouched instances, the fewer instances need to be considered during the investigation of raised alerts. Time spent on the investigation is then further reduced by prioritizing the order in which the instances are investigated according to their importance during the inference.

4.1 Algorithm Description

The algorithm for learning Instance Selection Randomized Trees (ISRT) follows the standard top-down greedy procedure for building an ensemble of unpruned decision trees. Every tree learner recursively divides the training sample set into two subsets until class homogeneity is reached or the samples can not be divided any further.

The main difference to the standard (single instance) tree-based learners applies in the way the conditions are evaluated inside the splitting nodes. In the MIL setting, the decision whether to send a sample (i.e. bag) to the left or right branch can no longer be based on a condition of type — *if feature* f *is greater than value* v (Equation 3.1) — as the bag might contain multiple feature vectors (i.e. instances) that may or may not fulfill that condition. To cope with this problem, every node of ISRT (denoted as $\mathcal{N}_{\text{ISRT}}$) is further parametrized with vector $\mathbf{w}$, called instance selector, in addition to the feature index f and the threshold value v (Equation 4.1). The

purpose of the instance selector is to select a single instance $\mathbf{x}^*$ from a bag $\mathcal{B}$ upon which the original feature value comparison $x_f^* > v$ is made. The selection mechanism is implemented via calculating the inner product (denoted as $\langle \cdot, \cdot \rangle$) between the vector $\mathbf{w}$ and individual bag instances $\mathbf{x} \in \mathcal{B}$, followed by selecting the instance $\mathbf{x}^*$ associated with the maximum response:

$$\mathcal{N}_{\text{ISRT}}(\mathcal{B}; \underbrace{f, v, \mathbf{w}}_{\Phi}) = \begin{cases} \text{left}, & \text{if } x_f^* > v, \ \mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{B}} \langle \mathbf{w}, \mathbf{x} \rangle, \\ \text{right}, & \text{otherwise.} \end{cases} \tag{4.1}$$

Note that if bags are of size one, then ISRT nodes behave like the traditional ones regardless of the extra parameter $\mathbf{w}$.

Assuming the positive class is the class of interest, we would like to train an instance selector (on a local training subset available to the considered node) to give maximum values to the instances of positive bags (and thus cause their selection) that are most responsible for these bags being positive. More specifically, the selector should assign low (i.e. negative) values to all instances of negative bags and high (i.e. positive) values to at least one instance from each positive bag. We do not force the selector to assign high values to all instances of positive bags, since not all of them are necessarily relevant. For example, if bags represent network users and instances visited websites of those users, not all websites visited by a malware-infected user within the last 24 hours are automatically malicious. In fact, the vast majority of them will typically still be legitimate. These requirements lead to the following zero-one loss function for a single training data point $(\mathcal{B}, y), \ y \in \{-1, 1\}$:

$$\ell_{01}\left(\mathbf{w}; (\mathcal{B}, y)\right) = \mathbb{1}\left[y \max_{\mathbf{x} \in \mathcal{B}} \langle \mathbf{w}, \mathbf{x} \rangle < 0 \right], \tag{4.2}$$

where $\mathbb{1}[\cdot]$ stands for an indicator function, which equals one if the argument is true and zero otherwise. If we approximate the indicator function $\mathbb{1}[z]$ with hinge loss surrogate $\mathrm{h}(z) = \max\{0, 1 - z\}$, take average over the local training subset $\mathcal{S} \subseteq \mathcal{D}$ and add regularization term λ, we obtain Multiple Instance Support Vector Machines [7] optimization problem:

$$\operatorname*{argmin}_{\mathbf{w}} \ \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{|\mathcal{S}|} \sum_{(\mathcal{B}, y) \in \mathcal{S}} \max\{0, 1 - y \max_{\mathbf{x} \in \mathcal{B}} \langle \mathbf{w}, \mathbf{x} \rangle\}. \tag{4.3}$$

To approximately solve this non-convex optimization problem in linear time, we adapted Pegasos solver [102] (originally designed for conventional SVMs) to the MIL setting. The resulting pseudo-code is given in Algorithm 2. It is a stochastic sub-gradient descent-based solver, which at each iteration t updates the current solution $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \eta^t \nabla^t$ (row 9 in Algorithm 2) using step size $\eta^t = 1/(t\lambda)$ and sub-gradient ∇^t of the objective function (Equation 4.3) estimated on a single randomly chosen training sample. To avoid building strong classifiers inside nodes, which would go against the randomization principle for constructing diverse independent trees [18], we restrict the selectors to operate on random low-dimensional subspaces. Input zero-one vector $\mathbf{s} \in \{0, 1\}^d$ then serves as a mask defining the feature subspace. By taking the element-wise product with that vector (i.e. $\mathbf{s} \odot \mathbf{w}$ or $\mathbf{s} \odot \mathbf{x}$), only feature positions occupied by ones remain effective. In Section 4.3, we empirically demonstrate that using sparse selectors, where the number of effective dimensions equals the square root of the total dimensions d rounded to the closest integer (i.e. $\sum_f s_f = [\sqrt{d}]$), plays a crucial role in the overall ensemble performance. This subspace size ensures that in high dimensions the selectors will be approximately orthogonal and thus independent [55].

Being equipped with the routine for training selectors, we can represent bags in a local training subset $\{(\mathcal{B}, y)_1, \ldots\}$ with selected instances $\{(\mathbf{x}^*, y)_1, \ldots\}$. Now, on top of this representation, a standard search for the best splitting parameters, based on measuring the purity of

Algorithm 2: ISRT's routine for training selectors.

Function TrainSelector($\mathcal{S}; \lambda, E, \mathbf{s}$)

Input : Training set of bags along with labels $\mathcal{S} = \{(\mathcal{B}, y)_1, \ldots\}$,
regularization $\lambda > 0$,
number of epochs $E > 0$,
zero-one vector defining feature subspace $\mathbf{s}$.

Output: Instance-level selector $\mathbf{w}^t$ approximately solving Problem 4.3.

extend all instances by a bias term $[\mathbf{x}, 1]$ *including the subspace vector* $[\mathbf{s}, 1]$
$t \leftarrow 1$
$\mathbf{w}^0 \leftarrow$ *random vector* $w_f^0 \sim N(0, 1)$ // $\text{length}(\mathbf{w}) = \text{length}(\mathbf{x}) = \text{length}(\mathbf{s})$
$\mathbf{w}^1 \leftarrow \mathbf{s} \odot \mathbf{w}^0$ // $\odot$ element-wise product
for 1 **in** E **do**
  for 1 **in** $|\mathcal{S}|$ **do**
    $(\mathcal{B}, y) \leftarrow$ *(class-balanced) random draw (with replacement) from* $\mathcal{S}$
    $\mathbf{x}^* \leftarrow \text{argmax}_{\mathbf{x} \in \mathcal{B}} \langle \mathbf{w}^t, \mathbf{x} \rangle$
    $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - \frac{1}{t\lambda} \left(\lambda \mathbf{w}^t - \mathbb{1} \left[y \langle \mathbf{w}^t, \mathbf{x}^* \rangle < 1 \right] y(\mathbf{s} \odot \mathbf{x}^*) \right)$
    $t \leftarrow t + 1$
return $\mathbf{w}^t[\text{start} : \text{end} - 1]$ // removing the bias term

produced subsets (e.g. Information gain [94] or Gini impurity [19]), can be executed. This allows us to build an ensemble of ISRT in the same way as (Extremely) Randomized Trees [47] are. In particular, unlike e.g. Breiman's Random Forests [18], where each tree is grown on a bootstrap replica of the training data, the Randomized Trees (as well as our BLRT and ISRT) are grown on the complete training set, which yields to a lower bias. Variance is then reduced by the output aggregation of more diversified trees. The higher diversification is achieved through stronger randomization in splitting nodes, as for each feature f out of $\lceil \sqrt{d} \rceil$ randomly selected ones, only a limited number[1] T of uniformly drawn threshold values v from $[x_f^{\min}, x_f^{\max})$ is considered for splitting rather than every sample value as realized in Breiman's Random Forests. In the case of ISRT, the randomization is even stronger due to the fact that selected instances may vary from node to node. The whole training procedure of ISRT is summarized in Algorithm 3.

From the computational viewpoint, the time complexity of the algorithm, assuming balanced trees, is $\Theta(M\sqrt{d}\, T\, E\, N_I \log N_{\mathcal{B}})$, where M is the number of constructed trees, E the number of epochs for training selectors, N_I the number of instances in bags and $N_{\mathcal{B}}$ the number of training bags.

In the testing phase, a bag to be classified is propagated through individual trees and the final score $\hat{y} \in [-1, 1]$ is calculated as an average of leaves scores the bag falls into (Algorithm 4). However, besides the prediction score, the ISRT can also output a histogram of selection counts over the bag instances $\mathbf{i}$. This information might help to identify relevant instances upon which the decision was made and thus serve as an explanation for positive bag predictions. For example, an explanation — *a user was found to be malware-infected because it has communicated with these three domains (out of hundreds) within the last 24 hours* — can greatly speed up the work of threat analysts and reinforce their trust in the model if the domains will be shown to be indeed malicious. On the other side, it should be noted that this approach can not explain a positive prediction that would be based on an absence of some type of instance(s) in the bag. For example, there might be malware, hypothetically, its only visible behaviour would be preventing an

[1] Term *extremely* in Extremely Randomized Trees [47] corresponds to setting $T = 1$.

Algorithm 3: ISRT's routine for building an ensemble of trees.

Function BuildTreeEnsemble($\mathcal{D}; M, T, E, \Lambda$)

Input : Training set of bags along with labels $\mathcal{D} = \{(\mathcal{B}, y)_1, \ldots\}$,
number of trees to grow $M > 0$,
number of considered threshold values $T > 0$,
number of epochs $E > 0$ (for training selectors).
Output: Ensemble of Instance Selection Randomized Trees $\mathcal{E}$.

$\mathcal{E} \leftarrow \emptyset$
for 1 **in** M **do**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\text{BuildTree}(\mathcal{D}; T, E, \Lambda)\}$
return $\mathcal{E}$

Function BuildTree($\mathcal{S}; T, E$)

Input : Local training subset $\mathcal{S} \subseteq \mathcal{D}$.
Output: Node with followers or Leaf with a prediction score.

if *all class labels y in $\mathcal{S}$ are equal* **then**
  return Leaf (y)
$\Phi^* \leftarrow$ FindBestSplittingParameters($\mathcal{S}; T, E$)
if $\Phi^* = \emptyset$ **then**
  return Leaf ($\frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} y$)
$\mathcal{S}_{\text{left}} \leftarrow \{(\mathcal{B}, y) \in \mathcal{S} \mid \mathcal{N}_{\text{ISRT}}(\mathcal{B}; \Phi^*) = \text{left}\}$
$\mathcal{S}_{\text{right}} \leftarrow \mathcal{S} \setminus \mathcal{S}_{\text{left}}$
if $\mathcal{S}_{\text{left}} = \emptyset$ *or* $\mathcal{S}_{\text{right}} = \emptyset$ **then**
  return Leaf ($\frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} y$)
return Node (Φ^*, BuildTree($\mathcal{S}_{\text{left}}$), BuildTree($\mathcal{S}_{\text{right}}$))

Function FindBestSplittingParameters($\mathcal{S}; T, E$)

Output: Triplet of splitting parameters Φ as defined in Equation 4.1.

$\Phi^* \leftarrow \emptyset$
$\mathbf{s} \leftarrow$ *zero-one vector of length d with* $\left\lceil \sqrt{d} \right\rceil$ *ones at random positions*
$\mathbf{w} \leftarrow$ TrainSelector($\mathcal{S}; \lambda = 1, E, \mathbf{s}$)
$(\mathbf{X}^*, \mathbf{y}) \leftarrow$ *represent each pair* $(\mathcal{B}, y) \in \mathcal{S}$ with $(\text{argmax}_{\mathbf{x} \in \mathcal{B}} \langle \mathbf{w}, \mathbf{x} \rangle, y)$
foreach *feature f* **in** $\left\lceil \sqrt{d} \right\rceil$ *randomly selected ones (without replacement) having non-constant values in* $\mathbf{X}$ *(i.e.* $x_f^{min} \neq x_f^{max}$*)* **do**
  foreach *value v* **in** T *uniformly drawn values from* $[x_f^{min}, x_f^{max})$ **do**
    $\Phi \leftarrow (f, v, \mathbf{w})$
    update $\Phi^* \leftarrow \Phi$ **if** Score($\Phi, \mathbf{X}^*, \mathbf{y}$) *is the best so far found score*
return Φ^*

operating system (or other applications like an anti-virus engine) from regular updates. For the same reason, negative predictions in general can not be explained with this approach.

Algorithm 4: ISRT's prediction routine.

Function Predict($\mathcal{B}; \mathcal{E}$)

Input : Bag to be classified $\mathcal{B}$ with ensemble of trees $\mathcal{E}$.
Output: Bag score $\hat{y} \in [-1, 1]$ and histogram of instance selection counts $\mathbf{i}$.

$\hat{y} \leftarrow 0$
$\mathbf{i} \leftarrow$ *zero vector of length* $|\mathcal{B}|$
foreach Tree **in** $\mathcal{E}$ **do**
  P $\leftarrow$ Tree `// pointer to Node and Leaf structures`
  $\mathbf{i}' \leftarrow$ *zero vector of length* $|\mathcal{B}|$
  while P *is of type* Node **do**
    $(b, i_{\mathbf{x}^*}) \leftarrow \mathcal{N}_{\text{ISRT}}(\mathcal{B}; \mathsf{P}.\Phi^*)$ `// ` $i_{\mathbf{x}^*}$ `index of selected instance`
    $\mathbf{i}'[i_{\mathbf{x}^*}] \leftarrow \mathbf{i}'[i_{\mathbf{x}^*}] + 1$
    P $\leftarrow \mathsf{P}.b$ `// continue in left or right branch`
  $\hat{y} \leftarrow \hat{y} + \mathsf{P}.y$
  $\mathbf{i} \leftarrow \mathbf{i} \oplus (\mathbf{i}' / \max(\sum_i \mathbf{i}'[i], 1))$ `//` $\oplus$ `element-wise addition`
$\hat{y} \leftarrow \hat{y}/|\mathcal{E}|$
$\mathbf{i} \leftarrow \mathbf{i} / \sum_i \mathbf{i}[i]$
return $(\hat{y}, \mathbf{i})$

4.2 Evaluation on Public Datasets

To verify the applicability of ISRT, we evaluate the algorithm on the 12 most popular MIL benchmark datasets that originate from six different domains[2]. Namely, classification of molecules (Musk1-2), classification of images (Fox, Tiger, Elephant), text categorization (Newsgroups1-3), protein binding site prediction (Protein), breast cancer detection (BreastCancer) and drug activity prediction (Mutagenesis1-2). Their basic meta-descriptions (i.e. counts of positive/negative bags, average bag size and feature dimension) are given in Table 4.1. For more details, we refer the reader to the survey of MIL datasets [28].

Since each dataset contains a predefined list of splitting indices for 5 times repeated 10-fold cross-validation, we strictly adhere to this evaluation protocol, just as we did while evaluating BLRT in Section 3.2. We also use similar hyper-parameter configurations. In particular, to train ISRT, we set the ensemble size to $M = 500$, the number of considered threshold values to $T = 8$ and the number of epochs for training selectors to $E = 1$. The BLRT model is trained with $M = 500$ and $T = 8$. In addition to BLRT and ISRT model, MI-SVM [7] and MIL-NN [91] are included in the evaluation as representatives of instance-level and bag-level methods, respectively. The MI-SVM model is trained using L2 regularization $\lambda = 10^{-3}$ and 100 epochs. The multiple instance neural network architecture then consists of a single instance layer of size 16 with rectified linear units (ReLU) followed by the `mean-max` aggregation and a single bag layer reducing the 32 units to the two output neurons. The weights are regularized with L1 regularization $\lambda = 10^{-3}$ to decrease overfitting as suggested in [91]. The training procedure minimizes a cross-entropy loss function using ADAM optimizer, mini-batch of size eight, and 1000 epochs at the maximum.

[2]Datasets are accessible at https://doi.org/10.6084/m9.figshare.6633983.v1.

Dataset	Bags +/-	Inst.	Feat.	MI-SVM	MIL-NN	BLRT ours	ISRT ours
Musk1	47/45	5	166	85.9 (1.9)	91.9 (1.5)	**96.8** (1.6)	**97.2** (1.3)
Musk2	39/63	65	166	86.9 (1.5)	**90.3** (2.3)	**91.2** (1.8)	**92.3** (2.6)
Fox	100/100	7	230	55.2 (1.6)	65.9 (1.2)	**73.3** (1.4)	**74.0** (1.8)
Tiger	100/100	6	230	81.7 (2.7)	**90.7** (1.7)	**92.6** (1.0)	**92.5** (0.8)
Elephant	100/100	7	230	84.5 (0.3)	**93.9** (0.8)	**95.8** (0.9)	**95.0** (0.7)
Newsgroups1	50/50	54	200	**82.4** (4.9)	77.0 (3.6)	**78.8** (2.6)	55.4 (2.6)
Newsgroups2	50/50	31	200	**70.2** (3.7)	**63.3** (5.2)	**63.0** (4.0)	**63.8** (2.9)
Newsgroups3	50/50	52	200	54.5 (5.5)	63.9 (4.1)	**76.3** (4.1)	65.0 (2.6)
Protein	25/168	138	9	81.2 (1.7)	75.2 (4.2)	74.9 (2.3)	**85.8** (2.0)
BreastCancer	26/32	35	708	73.1 (2.6)	**76.7** (8.1)	**84.5** (2.5)	79.3 (1.9)
Mutagenesis1	125/63	56	7	53.4 (1.0)	**90.2** (1.0)	**92.1** (1.3)	88.6 (0.8)
Mutagenesis2	13/29	51	7	70.0 (8.2)	66.2 (2.6)	**86.0** (3.5)	70.0 (6.6)

Table 4.1: Metadata about 12 public datasets. Including the number of positive/negative bags, the average number of instances inside bags and the number of features. Plus evaluation results, measured in AUC $\times$ 100, for individual models. Best results are shown in boldface. Multiple models are highlighted if the difference is not statistically significant (at $\alpha = 0.05$) according to a paired t-test with Holm-Bonferroni correction (for multiple comparisons) [32] computed on the five runs of 10-fold cross-validation.

Table 4.1 shows the performance of each model on each dataset in terms of the average Area Under the ROC Curve (AUC)[3] $\pm$ one standard deviation. It can be seen that the proposed ISRT model significantly outperforms the other three models only on Protein dataset, whereas the prior BLRT model significantly wins on two datasets (Newsgroup3 and Mutagenesis2). The average ranks[4] of the models are: BLRT=1.8, ISRT=2.0, MIL-NN=3.0 and MI-SVM=3.2. According to the non-parametric Friedman-Nemenyi test [32] (comparing all classifiers to each other based on the average ranks), there is no statistically significant difference[5] (at $\alpha = 0.05$) among the models, except for the pair BLRT and MI-SVM, where MI-SVM loses.

Analysis of instance-selection counts: In addition to a prediction score, ISRT also gives, for each classified bag, a histogram of selection frequencies[6] of individual bag instances. Figure 4.1 shows these selection frequencies for the ten most often selected instances from each bag averaged across all bags of a particular dataset. The non-uniformity of the values, especially on Newsgroups and Protein datasets, means that some instances are substantially more often selected than others. If it holds that the more frequently an instance is selected, the larger its impact on the final decision, then the selection counts could be a good proxy for ranking how relevant individual instances are to analysts validating model (bag-level) decisions.

Unfortunately, we can not assess the quality of this ranking criteria here, because annotations for instances are not provided with the used datasets. Nevertheless, we can study the worst-case scenario where the relevant instances would be among the least frequently selected ones. In such cases, analysts must go through all the instances that potentially affected a decision to find evidence for the raised alarm or to declare a false alarm. Going through all such instances in the ranked order corresponds to a cumulative sum approaching one on the ordered histogram of selection frequencies. Table 4.2 shows average percentages of bag instances that need to be analyzed to reach certain levels (e.g. 0.8, 0.9, 0.99) of the cumulative sum. For example, on the

[3] AUC is agnostic to class imbalance and classifier's decision threshold value.

[4] The best model is assigned the lowest rank (i.e. one).

[5] The performance of any two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference, which is (for 12 datasets, four methods and $\alpha = 0.05$) ~ 1.35.

[6] The histogram values (i.e. selection counts) are normalized to sum to one (Row 13 in Algorithm 4).

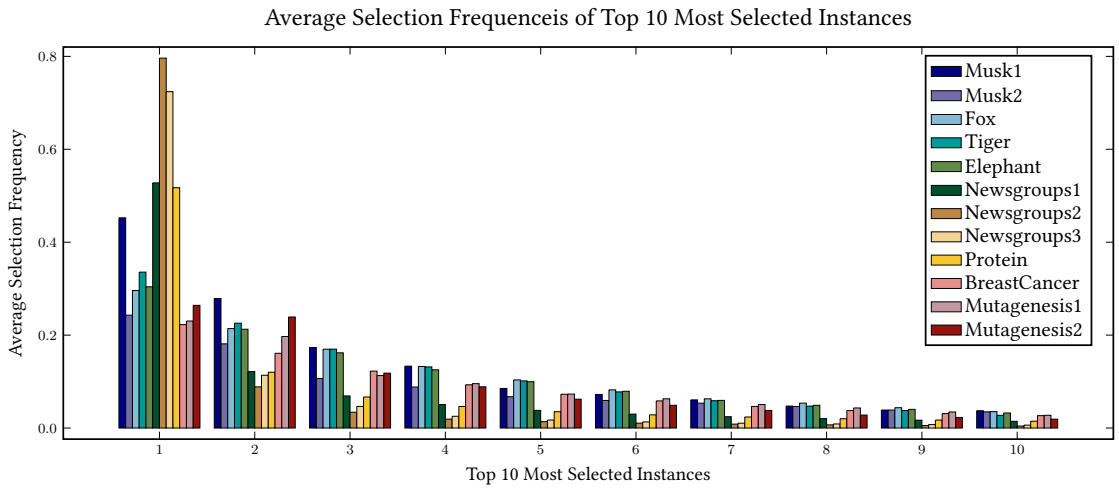

Figure 4.1: Ten highest values from the histogram of instance selection frequencies as produced by ISRT's prediction routine (Row 13 in Algorithm 4) averaged across all bags of a particular dataset. The rapid decrease of values in the case of Newsgroups1-3 and Protein datasets means that only a tiny fraction of instances are frequently selected and the rest is mainly ignored. This knowledge about the most often selected instances upon which the decision was made might considerably speed up any further investigation of positive bag predictions.

Protein dataset with the largest bag sizes (138 instances on average), only 21% of bag instances would need to be analyzed on average, since the remaining instances could not alter the model outcomes in any significant way (cumulative sum $= 0.99$). This already represents a substantial workload reduction for analysts, even under the worst-case scenario. If we admit a slightly more optimistic scenario that the relevant instances would be among the 80% of the most selected instances (i.e. cumulative sum $= 0.8$), then only 5% of bag instances need to be analyzed.

Dataset / CuSum	0.5	0.8	0.9	0.95	0.99	Avg. Inst.
Musk1	45% (2)	80% (3)	92% (4)	97% (5)	100% (5)	5
Musk2	34% (7)	61% (16)	73% (21)	82% (25)	90% (29)	65
Fox	42% (3)	73% (5)	87% (6)	95% (6)	99% (7)	7
Tiger	42% (2)	73% (4)	87% (5)	95% (6)	100% (6)	6
Elephant	39% (3)	70% (5)	85% (6)	94% (6)	99% (7)	7
Newsgroups1	3% (1)	10% (5)	18% (10)	27% (14)	41% (22)	54
Newsgroups2	4% (1)	5% (2)	10% (3)	18% (5)	36% (11)	31
Newsgroups3	2% (1)	4% (2)	9% (4)	15% (7)	29% (14)	52
Protein	**1% (1)**	**5% (6)**	**9% (11)**	**13% (17)**	**21% (26)**	**138**
BreastCancer	10% (4)	24% (8)	34% (12)	44% (15)	59% (20)	35
Mutagenesis1	6% (3)	14% (7)	18% (9)	22% (11)	27% (14)	56
Mutagenesis2	7% (3)	13% (6)	18% (8)	22% (10)	28% (13)	51

Table 4.2: Relative (and absolute) counts of top instances that analysts validating model decisions must process on average to reach a certain cumulative sum (e.g. 0.8, 0.9, 0.99) on the histogram of selection frequencies. On the Protein dataset, even under the worst-case scenario (i.e. ranking based on the selection frequencies is contrary to the best one), only 21% of bag instances would need to be analyzed on average, since the rest of the 79% instances were *almost* (CuSum $= 0.99$) untouched during the label inference.

4.3 Ablation Study

As the last experiment, we investigate the influence of individual model components and parameters on the final performance. Figure 4.2 shows results from this ablation study as a series of eight pair-wise comparisons. Each subplot compares two different variants (horizontal and vertical axis) of the proposed ISRT algorithm on the 12 datasets (dots on the scatter plot). X and Y coordinates of each dot are determined by the achieved AUCs of the corresponding variants on that particular dataset. Therefore, if a dot lies above the main diagonal, the variant associated with the vertical axis outperforms the other one associated with the horizontal axis and vice versa.

The first two **Subplots (A-B)** illustrate the effect of the ensemble size. While it is almost always better to build 100 trees than 5, building 500 trees usually does not bring any additional performance compared to 100. In **Subplot (C)**, we examine the model stability with respect to different random seeds (1234 vs. 42) and as can be seen, there is almost no difference. **Subplot (D)** shows a slight improvement that can be achieved by considering more thresholds for splitting ($T = 8$) than one as it is characteristic for Extremely Randomized Trees [47]. It is also worth experimenting with the sparse vs. dense selectors because, as can be observed from **Subplot (E)**, this option has different effects on different datasets. **Subplot (F)** then supports the idea that strongly randomized trees, unlike Breiman's Random Forests, do not have to be trained on the bootstrapped datasets. In **Subplot (G)** we analyze whether the search of splitting parameters $\Phi = (f, v, \mathbf{w})$ over multiple selectors with different regularization values $\lambda \in \{10^{-4}, 10^{-3}, \ldots, 1\}$ instead of one $\lambda = 1$ can help. Finally, the last **Subplot (H)** indicates that there is no need to train selectors with a large number of epochs.

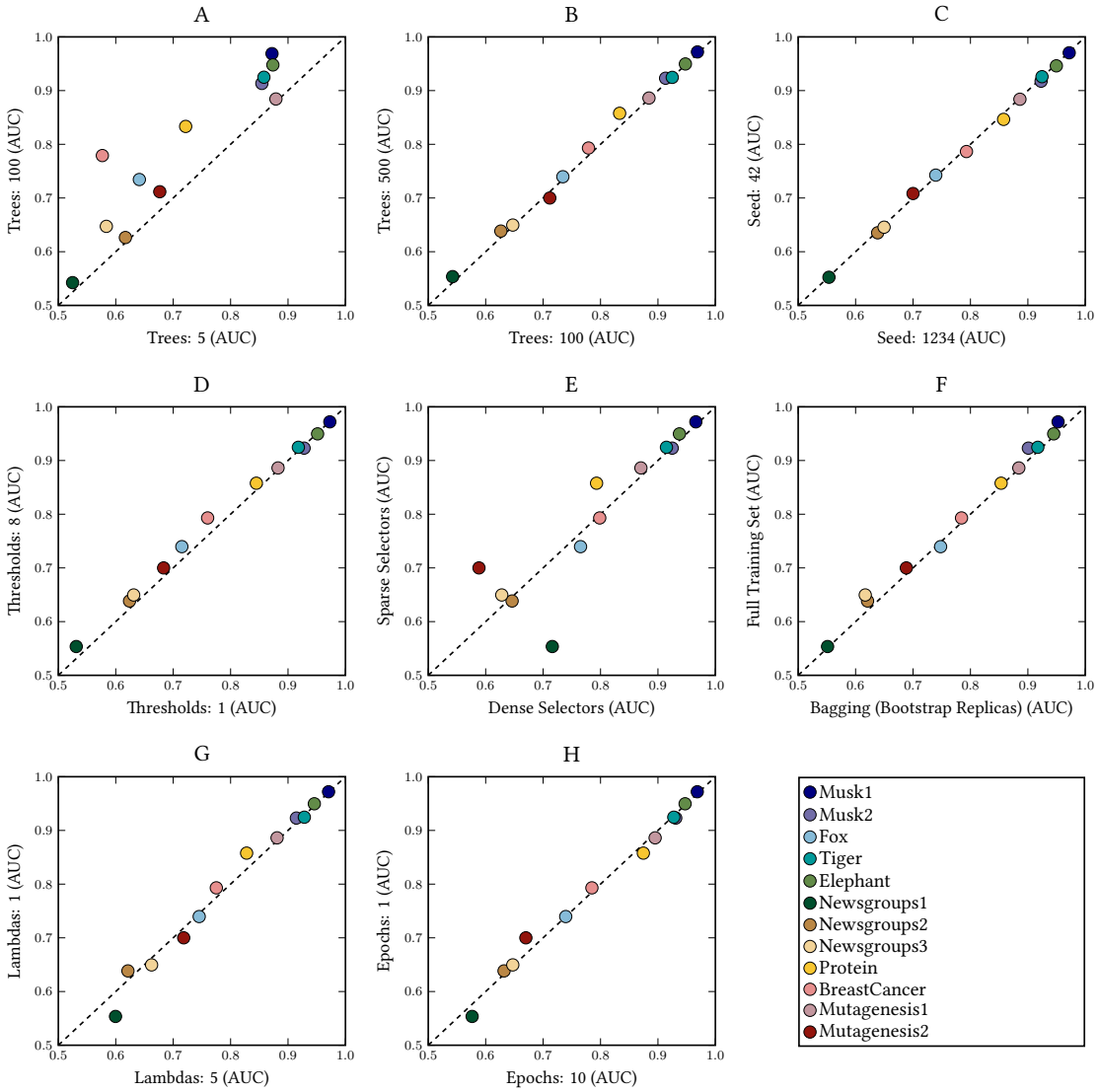

Figure 4.2: Ablation study assessing the influence of individual model components and parameters on 12 datasets. It illustrates the effect of the ensemble size **(A-B)**, the stability wrt. random seed **(C)**, the slight improvement caused by considering more thresholds for splitting **(D)**, the impact of sparse vs. dense selectors **(E)**, the no need for using bagging **(F)**, multiple regularization values **(G)**, nor a large number of epochs for training selectors **(H)**.

4.4 Summarizing Comments

We have proposed a novel tree-based algorithm called Instance Selection Randomized Trees (ISRT) for solving binary classification MIL problems. The algorithm naturally extends traditional randomized trees, since bags of size one are processed in the standard way by evaluating single feature value conditions at each node. However, when bags contain multiple instances, every node selects one instance from the bag, upon which the decision of whether to send the bag to the left or right branch is made. The instance selection mechanism is implemented in tree nodes via an additional vector of parameters that are optimized during the tree construction process.

Making decisions upon deliberately selected instances at each step is essential for the algorithm as this forces to extract patterns from relevant instances while leaving the irrelevant ones untouched. Analysts validating positive predictions in production may then focus only on the selected instances and disregard the remainder, resulting in a substantial decrease in workload on large bags. Although we could not evaluate the prioritization of instances based on their selection counts due to the non-existing instance-level labels on the public datasets, later in Section 6.1 (Figure 6.3), we demonstrate on a real-world problem from cybersecurity that this ranking criterion not only works well and further reduces the time spent on the investigation, but it also helps to discover new, previously unknown positive instances.

On the Protein dataset, we could see that ISRT outperformed BLRT significantly. Chapter 6 on applications of BLRT and ISRT in cybersecurity shows that this is more of the norm than the exception. We provide three reasons for why ISRT might be superior to BLRT even though ISRT decides based on information from fewer instances. The first two reasons assume that the unselected instances represent mainly a background noise that is often subject to change, especially in the presence of concept-drift [57]. The instance selection constraint then works as a ***regularization*** preventing from overfitting to the noisy background data by simply not learning from these instances. During the label inference, the same mechanism also works as a ***noise reduction filter*** eliminating the impact of newly incoming noisy instances on the decision-making process by non-selecting such instances. The third reason is ISRT's extended capability to extract ***local multivariate patterns*** from specific bag instances while BLRT can extract only *global univariate statistics* computed across all instances. As an illustration, BLRT cannot separate the following two bags: $\{(0,0),(1,1)\}$ and $\{(0,1),(1,0)\}$. Similarly, `max-mean-min-std` embedding methods [38, 49, 22] can't do that as the computed global statistics of both bags are indistinguishable from each other. ISRT achieves this by selecting specific bag instances (e.g. $(1,1)$ and $(1,0)$) multiple times and testing the necessary conditions on their feature values one by one (i.e. $x_1 > 0 \ \wedge \ x_2 > 0$). *Global patterns* are extracted in ISRT by accumulating knowledge during the sequential decision-making process. The ability to extract information from both ends of the spectrum, the local instance-level and the global bag-level, together with the regularization and input noise reduction mechanism, is most likely what will make ISRT excel in cybersecurity applications.

Chapter 5

JSON2bag Representation

This chapter introduces our JSON2bag method for converting arbitrary JavaScript Object Notation (JSON) documents to multiple instance bags, thereby enabling machine learning directly on raw JSON data.

JSON is a widely used data format designed to represent structured information in the form of a tree of arbitrary depth and breadth using recursive composition of objects and arrays with four primitive data types: string, number, boolean and null which makes it extremely useful for large amounts and types of data. However, despite being one of the most popular data transmission formats on the Internet [105], JSON is rarely used by machine learning practitioners. This is striking because JSON is strictly more expressive than, for example, the commonly used tabular Comma-Separated Values (CSV) format. In addition to a fixed-size set of attributes, JSON can easily accommodate further *optional* attributes, structured contextual information or knowledge about relationships with other entities without any pre-specified upper limit on their counts. And, although it is widely known that enriching the underlying data with such additional bits of available information can considerably improve models' prediction performance [4, 59, 61, 71], this practice is seldom realized via the JSON format.

We attribute this to the fact that there is no clear way how to derive models from non-vectorial JSON objects. Manual feature engineering is usually very time-consuming and error-prone because humans struggle to accurately estimate the importance of individual pieces of information in large volumes of complex data. Moreover, the variability in sizes of JSON arrays and objects between samples calls for using aggregation functions which might cause information loss. On top of that, any future modifications to the topology of processed JSON samples must be reflected in feature extractors accordingly, requiring additional human effort. Addressing the problem as sequence modeling [34] is also inappropriate due to the permutation invariances of key-value pairs in JSON objects and the diversity of JSON value data types. We therefore propose a simple yet effective MIL-based strategy for automated pattern extraction from raw JSON data. Results from an empirical evaluation show that the proposed approach is more robust to signal displacement in JSON structure and is applicable even to schema-less problems when compared with prior-art JSON-specific ML methods: JSON-NN [113] and JsonGrinder [122].

5.1 Algorithm Description

JSON2bag is a feature extraction algorithm for representing JSONs as sets of d-dimensional feature vectors (called bags of instances). This representation is MIL-compatible and allows learning models, such as BLRT (Chapter 3), ISRT (Chapter 4), and MIL-NN [91] to be built upon the converted JSON samples.

To handle the dynamic nature of the JSON format, JSON2bag takes advantage of the MIL paradigm, where the number of instances in bags is allowed to vary freely from bag to bag, just like the number of values accommodated inside individual JSON samples. Therefore, JSON2bag maps JSON values (i.e. leaves of the JSON trees) to instances. For this reason, the size of a bag representing a specific JSON sample equals the number of values stored in that sample. Since the location of a value in the JSON tree structure is essential as well, a respective instance (i.e. feature vector associated with that value) also encodes this information, in addition to the information about the value itself. As a result, every instance can be viewed as a *path-value* pair, where the path specifies the placement of a value in the JSON tree structure relative to the root node. Representing individual path-value pairs numerically as feature vectors (i.e. instances) is then far simpler to do without losing much information than trying to condense the whole JSON structure into a single feature vector.

Figure 5.1 illustrates the JSON2bag conversion as a two-step process. First, the tree structure of an input JSON sample is decomposed into a collection of path-value pairs, where the path is a sequence of object key names and array indices determining a position of a value (i.e. a string, a number, a Boolean or a null value[1]) with respect to the JSON root node. This process of decomposing hierarchically nested objects and arrays into a one-level structure is known as tree-flattening. Then, each path-value pair is converted into a feature vector by applying a battery of feature extractors to form an output bag (i.e. a set of feature vectors).

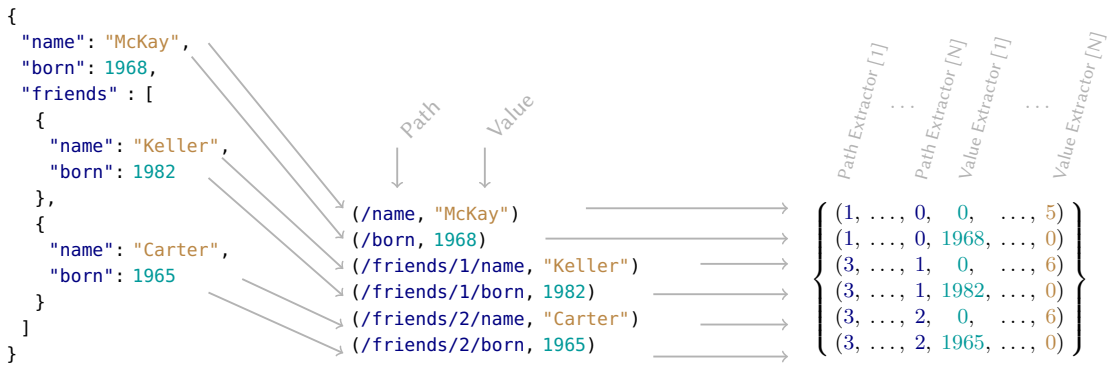

Figure 5.1: Two-step JSON2bag procedure for automated conversion of arbitrary JSON documents into sets of d-dimensional feature vectors (i.e. bags) suitable for MIL algorithms. The first step is to flatten an input JSON sample into a collection of path-value pairs. In the second step, a battery of path/value extractors extracts d features from each path-value pair to create an output set of feature vectors (i.e. a bag of instances).

There are two types of feature extractors: *path feature extractors* (encoding path characteristics like its length or the number of array indices in the path) and *value feature extractors* (encoding value properties like its type or the value itself if it is a number). A complete list of default extractors, including their parameter values (i.e. n-gram sizes and bucket counts for the hash-based extractors), is given in Table 5.1. They are designed to cover all common ways of using the JSON format so that they can be applied universally without needing to first go over the data. The aim is to preserve as much information as possible while keeping the feature dimensions reasonable. In total, 119 features are extracted from each path-value pair using this default battery of universal feature extractors with the default parameter values. This setting is used in all experiments in this thesis, unless stated otherwise.

[1] In the implementation, empty objects and arrays are also treated as values in order to distinguish the state of the present but empty container from the absence of the container.

Type	Features	Description
Path	1	Length of the path.
Path	1	Number of array indices in the path.
Path	1	Number of object key names in the path.
Path	1	Does the path terminate with an array index?
Path	1	Index of the first array in the path, or zero if none exists.
Path	1	Index of the last array in the path, or zero if none exists.
Path	32	Hashing individual object key names into 32 buckets.
Value	1	Is the value an empty array?
Value	1	Is the value an empty object?
Value	1	Is the value of type null?
Value	1	Is the value of type Boolean?
Value	1	Is the value of type Boolean and equals true?
Value	1	Is the value of type Number?
Value	1	Is the value of type Number and equals NaN?
Value	1	Is the value of type Number and equals positive infinity?
Value	1	Is the value of type Number and equals negative infinity?
Value	1	Value itself, if the value is a number.
Value	1	Value of $\log_2(\|\|x\|\| + 1)$, if the value is a number.
Value	1	Is the value of type String?
Value	1	Length of the string, if the value is a string.
Value	1	Ratio of digits, if the value is a string.
Value	1	Ratio of uppercase letters, if the value is a string.
Value	1	Ratio of punctual characters, if the value is a string.
Value	1	Ratio of non-ASCII characters, if the value is a string.
Value	32	Hashing n-grams (of size 3) of the string into 32 buckets.
Value	32	Hashing the lowercase version of the string into 32 buckets.

Table 5.1: List of path and value feature extractors including their feature ranges.

Note that potentially discriminative information can only be lost under two circumstances: when multiple object key names or string values fall into the same bucket, or when a path contains more than two array indices. The former case can be addressed by increasing the bucket counts and/or by using multiple hashing seeds and the latter by adding additional path extractors of type: "*Index of the N-th array in the path, or zero if none exists.*"

In the JSON2bag library, there are also available *non-universal* extractors that require some prior knowledge about the paths or values that can be encountered in a dataset of JSON samples. For example, one-hot encoding path/value extractors with predefined vocabularies of possible string values or object key names. Although such extractors are not among the default extractors, they are implemented in the library and can be included in order to tune the JSON2bag representation to the needs of a particular dataset. Furthermore, the existing battery of extractors can be easily expanded with additional custom implementations of (domain-specific) extractors, e.g., "*Is the string value a public or private IP address?*". Finally, the library also provides convenient ways for transforming and/or removing specific JSON values (e.g. unique identifiers, timestamps, class labels, etc.) based on their path or value type before converting the JSON samples to MIL bags.

5.2 Evaluation on Public Datasets

Firstly, we evaluate the proposed approach on Mutagenesis problem [31], for which a JSON-based representation[2], as well as a prior-art MIL-based representation[3], is publicly available. Our intention is, therefore, to compare the performance of models trained on the prior-art (task-specific) MIL representation with those trained on the MIL representation derived from the raw JSONs using our (generic) JSON2bag converter. The Mutagenesis problem is about classifying molecules trialed for mutagenicity on Salmonella typhimurium as either active (i.e. having positive log mutagenicity) or inactive (i.e. zero or negative log mutagenicity). Of the 188 molecules in the dataset, 125 are active and 63 inactive. Each molecule is described by the chemical properties of the entire molecule $\mathcal{M}$, its atoms $\{\mathcal{A}_1, \mathcal{A}_2, \ldots\}$ and the bonds between the atoms $\{b_{1,2}, b_{2,1}, \ldots\}$, as shown in Figure 5.2.

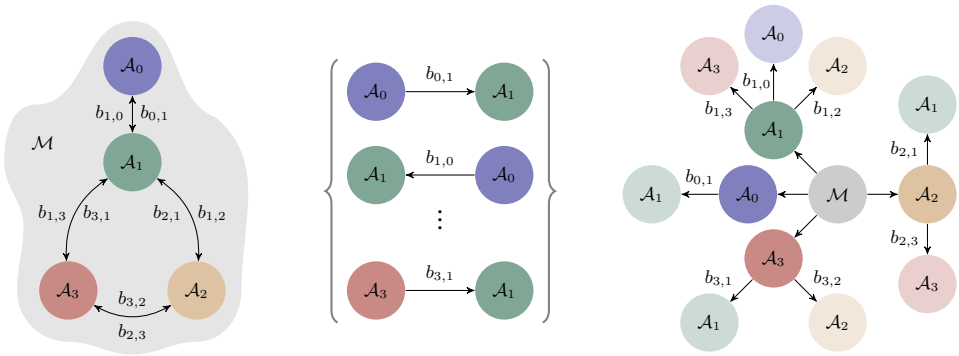

Figure 5.2: An illustration of a molecule from Mutagenesis dataset (on the left) and its two representations: the prior-art MIL representation (in the middle) and the JSON-based representation (on the right). The MIL-based representation simplifies the molecule graph structure into a set of atom pairs connected by a bond, whereas the JSON-based representation uses the JSON tree structure to semantically encode information about the molecule as a whole, its atoms and their adjacent neighbors.

In the prior-art MIL representation, each bag $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ corresponds to a molecule and each instance $\mathbf{x}$ to a pair of atoms in that molecule, which held together by a bond, i.e., instance $\mathbf{x} = (\mathcal{A}_1^1, \mathcal{A}_1^2, \mathcal{A}_1^3, b_{1,2}, \mathcal{A}_2^1, \mathcal{A}_2^2, \mathcal{A}_2^3)$, where $\mathcal{A}^f, f \in \{1, 2, 3\}$ is f-th atom feature. Example values of the three atom features (i.e. `"atom_type"`, `"element"` and `"charge"`) and the one bond feature (i.e. `"bond_type"`) are given in Figure 5.3 showing a molecule represented in the JSON format. Apparently, the prior-art MIL representation fails to capture the molecule-level properties (i.e. the missing `"lumo"`, `"logp"`, `"ind1"` and `"inda"` features when compared with the JSON-based representation) and reduces the complex molecule graph structure to just atom pairs. On the other hand, Figure 5.3 shows how the JSON-based representation semantically integrates information from all three levels of the structure (i.e. molecule, atoms and bonds) into a single JSON object representing a whole molecule. At the top level of the JSON hierarchy, there are molecule-related attributes; at the second level, there are attributes of individual molecule's atoms; and at the last third level, there are attributes of bonds and connected atoms. However, because JSON is a tree-structured format, and hence cannot fully reflect the structure of general graphs with possible loops, which are typical for the molecules, the JSON-based representation only stores information about a 1-step neighborhood of each molecule's atom. A notable side effect of both representations is the redundancy of atom-level attributes, as a single atom might be a neighbor of multiple atoms, which causes its features to appear numerous times in different

[2] https://juliaml.github.io/MLDatasets.jl/stable/datasets/misc/#MLDatasets.Mutagenesis
[3] https://doi.org/10.6084/m9.figshare.6633983.v1

parts of the representations. Figure 5.2 also illustrates both explored representations in addition to the molecule structure.

```
{
  "lumo": -1.246,
  "logp": 4.23,
  "ind1": 1,
  "inda": 0,
  "atoms": [
    {
      "atom_type": 3,
      "element": "h",
      "charge": 0.142,
      "bonds": [
        {
          "atom_type": 22,
          "element": "c",
          "charge": -0.117,
          "bond_type": 1
        }
        /... other bonds /
      ]
    }
    /... other atoms /
  ]
}
```

Figure 5.3: JSON-based representation of a molecule in the Mutagenesis problem.

While an average bag in the prior-art MIL representation has 56 instances of size 7 features (three atom features two times plus one bond feature), an average bag in the MIL representation obtained by converting the JSON objects with our JSON2bag technique has 305 instances of size 119 features. We assess the quality of both representations by measuring the prediction performance of three MIL models, BLRT (Chapter 3), ISRT (Chapter 4) and MIL-NN [91], trained on both types of bags. To train BLRT, we set the number of examined feature threshold values to 8 and the ensemble size to 100. The ISRT model is trained with identical parameter settings as BLRT plus the number of epochs for training instance selectors is set to 10. The MIL-NN model architecture consists of a single instance layer of size 32 with `mean-max` aggregation followed by a bag layer reducing the 64 rectified linear units (ReLU) to the two output neurons. The network weights are optimized via AdaBelief [120] optimizer with default values and mini-batches of size 10 to minimize cross-entropy loss with L1 regularization $\lambda = 10^{-3}$ for 100 epochs. We also evaluate the performance of two more ML algorithms designed specifically for JSON data: JsonGrinder [122] coupled with HMIL-NN [80] and JSON-NN [113]. JsonGrinder (JG) converts JSON data to *hierarchical* MIL bags that can be subsequently processed by Hierarchical MIL Neural Networks (HMIL-NN). To create the HMIL-NN model reflecting the JSON schema of molecules, we set the size of the inner dense layers to 10 and used AdaBelief optimizer with mini-batches of size 10 to optimize the network weights for 100 epochs. JSON-NN denotes Long Short-Term Memory (LSTM) [56] based implementation of Semantic Tree-structure Recursive Learning Algorithm (STRLA) adapted to JSON format [113] that we trained using the hidden layers of size 32, ADAM optimizer with the learning rate 0.01, mini-batches of size 10 and the epoch count set to 10[4]. Due to the small dataset size, all evaluations are conducted using the 5 times repeated 10-fold cross-validation schema, as mostly done in other studies dealing with the Mutagenesis dataset [77].

[4]Due to the JSON-NN's high computing needs, we reduced the number of epochs to 10 (from 100). While BLRT and ISRT were evaluated in less than five minutes, the evaluation of JSON-NN was completed in more than six hours, using the same computational power.

	Mutagenesis problem			
	Prior-art	JSONs		
	MIL features	Raw JSONs	JsonGrinder	JSON2bag [ours]
ISRT [ours]	88.7 (0.9)	×	×	**94.9** (0.9)
BLRT [ours]	92.5 (1.1)	×	×	**95.3** (1.1)
MIL-NN	90.6 (0.3)	×	×	**94.1** (1.0)
HMIL-NN	×	×	**94.1** (1.6)	×
JSON-NN	×	**94.4** (0.4)	×	×

Table 5.2: Classification performance (measured in AUC $\times$ 100) of models evaluated on two different representations of the same Mutagenesis problem. Models built on top of the JSON-based representation outperform those trained on the prior-art MIL features.

Table 5.2 shows results from the evaluation in terms of the average Area Under the ROC Curve (AUC)[5] $\pm$ one standard deviation. It can be observed that although there is no significant difference between the models trained on JSONs, they clearly outperform those trained on the prior-art MIL representation. The ISRT model exhibits the highest relative improvement of 7% when trained on converted JSONs with JSON2bag instead of the prior-art MIL representation, which was used before (e.g. in Section 4.2). Surprisingly, the performance of BLRT and ISRT models trained on converted JSONs remains significantly superior even when we remove the molecule-level properties (missing in the prior-art MIL representation) from the JSONs in order to balance the amount of captured information in both representations. In particular, after removing the molecule-level attributes (i.e. `"lumo"`, `"logp"`, `"ind1"` and `"inda"`), the performance of BLRT is $(95.2 \pm 1.1)\%$ and of ISRT is $(93.7 \pm 1.5)\%$. The classifiers, therefore, appear to generalize better in the higher dimensional MIL feature space induced by the proposed JSON2bag method (on average, 305 119-dimensional instances per bag/molecule) than in the lower-dimensional prior-art MIL feature space (on average, 56 7-dimensional instances per bag/molecule).

Robustness towards JSON distortions: In the next experiment, we analyze the resilience of the JSON methods (i.e. JSON-NN, JsonGrinder and JSON2bag) against two types of data distortions on the Mutagenesis dataset. In the first test, each training JSON sample (denoted as $\{\text{signal}\}$) is polluted with a randomly chosen negative/inactive sample (denoted as $\{\text{noise}\}$) to create a new noisy sample: $\{\text{a} : \{\text{signal}\}, \text{b} : \{\text{noise}\}\}$. This type of distortion simulates adding irrelevant background data to the JSON representation that does not alter the class label. While the first test applies the same distortion for both the training and the testing set, the second test additionally swaps the signal with the noise for the testing set: $\{\text{a} : \{\text{noise}\}, \text{b} : \{\text{signal}\}\}$. This second type of distortion simulates signal displacement within a JSON structure. Being able to correctly classify JSON samples under this type of distortion can be advantageous, particularly in cybersecurity, where the displacement of a (malicious) signal can be viewed as a detection evasion technique [3]. Table 5.3 summarizes results from the experiments, including the reference values for comparison with no data distortion.

As can be seen from Table 5.3, all approaches are able to cope with the added noise. When compared to the reference performance of each method (i.e. the first and the second column of the results), there are only statistically non-significant differences. In the case of the JSON2bag + ISRT pair, there is almost no difference: $(94.9 \pm 0.9)\%$ vs. $(94.9 \pm 0.7)\%$. Specifically, ISRT on the noisy dataset still selects just a small fraction of instances, 29 on average (instead of 26), despite the fact that there are now 526 instances per bag on average (instead of 305) due to the added noise. This means that most of the time, the added irrelevant instances are successfully

[5] AUC is agnostic to class imbalance and classifier's decision threshold value.

ignored. The second test (signal displacement), on the other hand, is only passed by the proposed JSON2bag method as the prediction performance of JSON-NN and JsonGrinder drops to the level of random guessing (i.e. AUC around 50%). This demonstrates that the prior-art approaches are oversensitive to the signal position within the JSON structure. In the case of the JsonGrinder + HMIL-NN pair, this is expected behavior because the *hierarchical* network architecture mirrors the JSON tree structure, so values in different leaves of the JSON tree are processed by different network layers[6]. Therefore, a layer cannot detect a (positive) signal during testing if it never saw it during training (since it was somewhere else in the JSON tree). In contrast, *non*-hierarchical MIL-NN built on top of the bags produced by JSON2bag (AUC of $(88.7 \pm 0.8)\%$) processes every bag instance (i.e. JSON path-value pair) using the same network weights optimized on all training samples. Therefore, changes in the position of a signal are reflected only in the feature values of respective instances and not in the route the signal goes through the network. Moreover, based on how JSON2bag works, a small change in the signal position (e.g. a different object key name: `"b"` instead of `"a"`) affects only a small quantity of *path* feature values of respective instances. This is probably why BLRT compares favorably to other methods (AUC of $(93.8 \pm 0.9)\%$), as each tree in the forest usually considers only a subset of features, which limits the impact of the affected features on the ensemble outcomes. ISRT then performs slightly worse (AUC of $(90.1 \pm 0.7)\%$) because the selectors in tree nodes consider a larger subset of features during the prediction (by default, each selector looks at the square root of the number of features), causing the affected features to have a higher impact on the final decision than in the case of BLRT.

	Mutagenesis problem with added distortions		
Training	{signal}	{a : {signal}, b : {noise}}	
Testing	{signal}	{a : {signal}, b : {noise}}	{a : {noise}, b : {signal}}
J2b $^{\text{ours}}$ + ISRT $^{\text{ours}}$	94.9 (0.9)	94.9 (0.7)	90.1 (0.7)
J2b $^{\text{ours}}$ + BLRT $^{\text{ours}}$	95.3 (1.1)	94.1 (0.6)	**93.8** (0.9)
J2b $^{\text{ours}}$ + MIL-NN	94.1 (1.0)	92.8 (1.1)	88.7 (0.8)
JG + HMIL-NN	94.1 (1.6)	92.4 (1.6)	47.4 (9.8)
JSON-NN	94.4 (0.4)	93.7 (1.1)	51.7 (7.4)

Table 5.3: Assessment of the robustness of JSON-based methods against two types of data distortions. The first column shows reference values (no distortion); the second column results when JSON samples are polluted with irrelevant noise, and the last column shows when the signal and the noise are swapped in the testing set. All techniques can handle the additional noise, but only JSON2bag can handle the signal displacement.

Modeling schema-less JSONs: Finally, we investigate the applicability of the JSON-specific ML methods to the problem of categorizing generic semi-structured documents. For this task, we use a publicly available Industry Sector dataset from http://www.iesl.cs.umass.edu/datasets.html consisting of 9,569 corporate web pages classified into 12 top-level categories, such as basic materials, capital goods, conglomerates, consumer cyclical and non-cyclical sectors, etc. See Table 5.4 for the complete list of the categories. We split the dataset into 6,048 training and 3,521 testing web pages. Prior to applying the ML methods, we convert the HTML source code of each web page into the JSON format, using the `html-to-json` Python package from https://pypi.org/project/html-to-json/. This conversion should preserve all information about the pages' content, structure, and style formatting. The average complexity of the resulting JSONs representing the web pages is 180 leaves (i.e. values) of depth (i.e. path length) 17.

[6]This is true unless the values are elements of the same array.

To create a numerical representation of the JSON samples, we employ JSON2bag. Since the JSONs contain mostly general (non-categorical) string values, we adjust the default parameter settings of JSON2bag's feature extractors as follows. The number of buckets for the n-gram hashing is increased to 64 (from 32), and the number of buckets for the whole string hashing is decreased to 16 (from 32). As a result, 135 features are extracted from each path-value pair to form one bag instance. We train BLRT and ISRT models of size 64 trees, using the default eight trials for feature thresholding and 50 epochs for fitting selectors in the case of ISRT. The multi-class problem is addressed via the one-vs-rest strategy, where one model per class is trained. We also train MIL-NN consisting of a single hidden layer of size 200 neurons with ReLU activation functions followed by `mean-max` aggregation and an output layer reducing the 400 units to 12 output (class) neurons. The network weights are optimized using the AdaBelief optimizer with mini-batches of size 100 for 200 epochs. The technique based on JsonGrinder and HMIL-NN is inapplicable to this task because the web pages (and the corresponding JSONs) do not follow any common schema that could be reflected in the network architecture. In other words, for every web page, it is possible to be structured differently. Unfortunately, neither the JSON-NN method is evaluated here because of its high computational demands for processing complex data structures such as those of web pages[7]. Though, unlike HMIL-NN, it does not require all JSONs to obey a fixed schema. Therefore, to have a baseline for comparison, we evaluated the Random Forest classifier trained on n-grams that are built from a textual representation of the web pages provided by the `html2text` Python package available at https://pypi.org/project/html2text/. Identical to the feature extractors of JSON2bag, n-grams of three (trigrams) are used. Each web page's n-grams are hashed into 1024 buckets to create feature vectors of the same length. Table 5.4 then gives a summary of outcomes from the evaluations.

From Table 5.4, it can be observed that the best result in terms of the accuracy over all classes (78.10%) is achieved by the ISRT model trained on the bags obtained by the proposed JSON2bag converter. MIL-NN (also trained on these bags) gives a comparable overall result (76.85%) and better per-class results for half of the classes. When compared with the overall accuracy (64.75%) of the baseline model (i.e. Random Forest trained on n-grams), it is evident that the JSON-based representation preserving the semantic structure and the formatting styles of pages is more advantageous than the plain textual representation. The inferior accuracy of BLRT (55.15%) is likely attributable to the use of non-specific *global* statistics (computed across all instances) in splitting nodes of BLRT (discussed in Section 4.4)[8]. Note that a dummy model always predicting the most frequent class (i.e. Services) would achieve the accuracy of 28.09%.

Explainability of JSON2bag and ISRT: A great advantage of ISRT is its native ability to provide explanations of model bag-level decisions in the form of the top-k most selected instances (Algorithm 4). In conjunction with JSON2bag, where instances correspond to JSON path-value pairs, this gives analysts easily comprehensible insights into model decisions upon individual JSONs. For the qualitative assessment, Table 5.5 shows examples of such explanations on the current task of categorizing corporate web pages. From each class (i.e. industry sector), one correctly classified JSON sample is chosen, and its top-k most selected path-value pairs (i.e. instances) are shown. As can be seen from the table, among the most selected path-value pairs, there are page titles, background images and colors, text and link colors. Such behavior makes sense since these values are typically the same for all pages of a company and different from pages of other companies. This also indicates that the model learned to recognize identities of

[7] One forward pass of the dataset through JSON-NN took more than four hours. According to the method description: "*The current iteration of this code is highly unoptimised, and hence is very slow for complex data structures*". Available at: https://github.com/EndingCredits/json2vec (accessed: 2022-12-24).

[8] Unlike the previous experiment with synthetically added noise, here, the ratio and the nature of noisy instances inside bags vary dramatically due to the substantial structural differences among the JSON samples (i.e. source codes of web pages). Therefore, taking into account every instance during the decision-making like in BLRT is misleading, and some instance selection mechanism in models is needed. Both ISRT and MIL-NN use the maximum aggregation function in tree nodes (Equation 4.1) or network neurons to retain just the signals of desired instances.

	Web page classification problem					
	HTML2TEXT	HTML2JSON				
	N-grams	JSON	JsonGrinder	JSON2bag [ours]		
	RF	JSON-NN	HMIL-NN	MIL-NN	BLRT [ours]	ISRT [ours]
Basic materials	68.63	?	—	76.41	42.90	**79.36**
Capital goods	51.43	?	—	69.80	40.41	**70.61**
Conglomerates	22.86	?	—	**42.86**	22.86	40.00
C. cyclical	52.52	?	—	72.41	45.89	**73.74**
C. non-cyclical	52.88	?	—	73.82	50.26	**80.10**
Energy	37.12	?	—	73.48	40.91	**74.24**
Financial	49.13	?	—	**70.35**	40.99	65.70
Healthcare	27.39	?	—	**65.61**	32.48	57.96
Services	96.06	?	—	87.06	89.79	**97.88**
Technology	63.83	?	—	**80.10**	41.99	68.69
Transportation	50.00	?	—	**78.31**	42.17	72.29
Utilities	34.00	?	—	**58.00**	29.00	50.00
Overall	64.75	?	—	76.85	55.15	**78.10**

Table 5.4: Accuracy of the baseline (Random Forest trained on n-grams) and JSON-specific methods on the web page classification problem with 12 classes. The JSON-based representation (produced by HTML2JSON) preserves not only the textual content of web pages (like HTML2TEXT) but also their semantic structure and formatting styles. This seems to be advantageous since JSON2bag in conjunction with ISRT yields the highest overall accuracy of 78.10%. Results for JSON-NN are unknown (denoted with symbol "?") due to the method's excessive time requirements on such complex data. The JsonGrinder + HMIL-NN technique is inapplicable since the structure of pages does not follow a single schema but varies from page to page. A dummy model always predicting the majority class (Services) would have an accuracy of 28.09%.

web pages of individual companies rather than characteristics of entire sectors, even though keywords like `"bank"` (Financial sector), `"oil"` (Energy sector), `"hardware"` (Technology sector), `"pharmacy"` (Healthcare sector), etc., in the titles, may generalize beyond the particular companies. Moreover, this also supports the above justification of the superiority of ISRT and MIL-NN over the baseline Random Forest as the missing information about the colors, backgrounds, and other formatting styles in the textual representation of web pages, indeed, seems to be crucial. Insights like these, in general, may not only facilitate the validation of model predictions but also provide a deeper understanding of the importance of individual pieces of information in complex data structures.

Class	Score	Top k most selected bag instances (i.e. JSON path - value pairs)
Basic materials	0.14	/html/1/head/1/**title**/1/_value/ Intertape Polymer Group
	0.13	/html/1/body/1/_attributes/**bgcolor**/ white
	0.11	/html/1/body/1/_attributes/**link**/ #004080
Capital goods	0.23	/html/1/head/1/**title**/1/_value/ New Holland - Asia Pacific
	0.14	/html/1/head/1/head/1/body/1/center/1/br/1/ EmptyObject()
	0.13	/html/1/head/1/head/1/body/1/_attributes/**background**/grafica/ wall3.gif
Conglomerates	0.19	/html/1/body/1/_attributes/**bgcolor**/ #FFFFFF
	0.12	/html/1/body/1/_attributes/**link**/ #2E26B2
	0.10	/html/1/**title**/1/_value/ Stone & Webster, Commercial Cold Storage
C. cyclical	0.22	/html/1/body/1/_attributes/**background**/img/ backgrnd.gif
	0.11	/html/1/body/1/table/1/tr/1/td/1/br/2/ EmptyObject()
	0.08	/html/1/head/1/**title**/1/_value/ Goodyear North American Tires
C. non-cyclical	0.18	/html/1/head/1/**title**/1/_value/ Te-Amo
	0.17	/html/1/body/1/_attributes/**vlink**/ #728400
	0.08	/html/1/body/1/_attributes/**link**/ #00007a
Energy	0.14	/html/1/head/1/**title**/1/_value/ Pennzoil
	0.10	/html/1/body/1/_attributes/**alink**/ #009900
	0.08	/html/1/body/1/_attributes/**link**/ #FF0000
Financial	0.24	/head/1/**title**/1/_value/ Summit Bank
	0.23	/frameset/1/_attributes/rows/ 75,*
	0.14	/frameset/1/_attributes/frameborder/ NO
Healthcare	0.10	/html/1/body/1/_attributes/**background**/ sky.jpg
	0.10	/html/1/body/1/_attributes/**bgcolor**/ #CCCCFF
	0.08	/html/1/head/1/**title**/1/_value/ Vitalink Pharmacy Services...
Services	0.23	/html/1/frameset/1/_attributes/cols/ 19%,81%
	0.14	/html/1/head/1/**title**/1/_value/ Attorney Murthy : U.S. Immigration Law
	0.12	/html/1/head/1/meta/1/_attributes/name/ GENERATOR
Technology	0.25	/html/1/body/1/_attributes/**bgcolor**/ #FFFFFF
	0.11	/html/1/head/1/**title**/1/_value/ Junior Hardware Engineer
	0.10	/html/1/head/1/meta/1/_attributes/name/ GENERATOR
Transportation	0.16	/html/1/body/1/_attributes/**bgcolor**/ #000000
	0.13	/html/1/body/1/_attributes/**text**/ #BDB9AD
	0.09	/html/1/body/1/_attributes/**vlink**/ #9425E6
	0.06	/html/1/body/1/_attributes/**link**/ #DE5208
	0.03	/html/1/head/1/**title**/1/_value/ FedEx Dropoff Locations...
Utilities	0.27	/html/1/body/1/_attributes/**bgcolor**/ #F9EAD0
	0.11	/html/1/body/1/_attributes/**background**/../images/ bk2.gif
	0.08	/html/1/head/1/**title**/1/_value/ National Fuel - Utility Operations

Table 5.5: Explainability of ISRT trained to categorize corporate web pages into 12 classes (industry sectors). For each class, one correctly classified web page (JSON/bag) is picked, and its top k most selected bag instances (i.e. JSON path-value pairs) including their selection frequencies (denoted as Score) are displayed. Apparently, the model decides mostly based on the title (usually containing the company name) and colors of texts/links/backgrounds which are typically shared across all company's pages and, at the same time, distinct from other companies' pages.

5.3 Comparing JSON2bag to JsonGrinder

When comparing JSON2bag to state-of-the-art JsonGrinder [122], the key difference lies in the structure of produced numerical representations. While JSON2bag produces *non-hierarchical* (one-level) bags following the traditional MIL paradigm [5] (Section 2.1), where samples are defined as sets of feature vectors, JsonGrinder follows the Hierarchical Multiple Instance Learning (HMIL) paradigm [89] (Section 2.2), where HMIL samples are defined as arbitrarily nested compositions of bag and product structures.

Contrary to non-hierarchical JSON2bag, the *hierarchical* JsonGrinder approach requires JSON samples to obey a fixed schema that serves as a structural template for the HMIL samples and the corresponding model (i.e. HMIL-NN [80]), giving rise to the following drawbacks:

1. Not every dataset of JSONs adheres to a specific schema. A prime example of this is the Industry Sector dataset of web pages discussed in Section 5.2, where each page may have its own unique HTML structure.
2. The hierarchical approach is designed to support only homogeneous JSON arrays, where elements share the same type or structure. This is because uniformity in the numerical representation across all array elements facilitates their subsequent aggregation within the bag structure that is used to model JSON arrays in JsonGrinder.
3. In every JSON object, key names must originate from a finite super-set and cannot be used to store arbitrary string values. Although using key names as values is generally considered poor practice, one does not always have control over the JSONs she/he has to work with.
4. As dictated by JSON schema, every dataset sample must maintain the same type of values for object key names located at particular positions within the particular JSON topology.
5. To the best of our knowledge, there is only one specific type of model (i.e. HMIL-NN [80]) that can be trained on top of the HMIL data samples.
6. Model complexity grows with schema complexity, regardless of the simplicity or complexity of the discriminative signal. Consequently, one might end up with a complicated model even when a simple discriminatory rule exists[9].
7. Since separate network layers process JSON values from different sub-parts of the schema, the overall model is not immune to shifts in signal locations as demonstrated in Section 5.2.

On the other hand, the *non-hierarchical* approach used by JSON2bag generates representations of considerably higher dimensions. Every instance encoding a JSON leaf value also encodes information about its location (i.e. the path from the root node to the leaf), which is unnecessary for the HMIL data samples as the location is reflected in the sample hierarchy. Moreover, every JSON leaf value is encoded using a complete set of features instead of a specialized subset based on the value type specific to the location within the schema.

5.4 Summarizing Comments

In this chapter, we have proposed a JSON2bag technique for automated end-to-end model learning on JSON data. JSON2bag converts every JSON sample into multiple feature vectors (called bag of instances) rather than a single feature vector. Such representation is compatible with MIL models and facilitates automated feature extraction, as every JSON can be decomposed into multiple path-value pairs that are encodable as feature vectors without losing much information. To this end, a battery of universal path/value feature extractors is designed. By extracting a fixed

[9] With JSON2bag and ISRT, we expect decision trees to reach only shallow depths, and the discriminative JSON value to appear among the most frequently selected instances.

set of features from each path-value pair of a JSON sample, a bag of instances representing that sample is created.

Using the JSON format and JSON2bag, we addressed two problems with complex data structures to demonstrate the applicability of this approach. The first task is about classifying graph-structured molecules where nodes are atoms and edges are chemical bonds connecting the atoms. Despite being a tree-structured format (unable to represent cyclic graphs completely), the JSON-based representation describes molecules as JSON objects containing arrays of atoms and their 1-step neighbors. Remarkably, all properties of atoms, molecules, and even bonds are taken into account with minimal work. Compared to the prior-art MIL representation [28], the JSON-based representation renders the task significantly easier to tackle for all employed MIL models. The second task is to categorize web pages, which is complicated by the fact that each page may have its own unique HTML structure. We demonstrate that JSON2bag operates even in these challenging schema-less settings (as the only technique) and, by factoring in the structural information, enhances the prediction performance of models.

Providing explanations may be necessary for models to be deployed in the wild, as it helps to validate model verdicts, identify possible bias or gain insights into data. JSON2bag in conjunction with ISRT gives explanations in the form of the top most often selected (anticipated to be the most important) JSON path-value pairs, as shown in Table 5.5 on the web page classification problem. In study [92], the explainability of JsonGridner and HMIL-NN is investigated. It has the form of a minimal subset of the JSON sample that is still classified to the same class as the complete sample. While it is currently beyond the scope of this work, exploring and comparing these two approaches both qualitatively and quantitatively presents an interesting opportunity for future research.

In conclusion, the proposed JSON2bag producing a one-level MIL representation of JSONs strikes a practical balance between two extremes: on one hand, the traditional non-MIL single-feature vector human-defined representation, which, despite being popular, is challenging to design without significant information loss, and on the other hand, the multi-level HMIL representation provided by JsonGrinder, which, while exhaustive [89], suffers from several drawbacks (Section 5.3) due to the requirement of a JSON schema defining the hierarchical multi-level structure. This proposed JSON2bag approach does come with the trade-off of higher sample dimensionality, but we believe this to be a worthwhile cost given the benefits in other areas.

Chapter 6

Applications to Cybersecurity

In this chapter, we address five cybersecurity problems using the techniques developed in the previous three chapters. The aim is to demonstrate the usefulness and the broad applicability of the proposed tools on a variety of challenging real-world problems.

The first problem, *Infected User Detection Based on HTTP Logs* (Section 6.1), is formulated as a MIL task and addressed by BLRT (Chapter 3) and ISRT (Chapter 4) models, whereas the next four problems, *Infected User Detection Based on Network Events* (Section 6.2), *Malicious Domain Discovery* (Section 6.3), *Malicious Email Detection* (Section 6.4), and *Malicious Executable Classification* (Section 6.5), are formulated as a JSON classification and addressed by JSON2bag (Chapter 5) in conjunction with BLRT and ISRT models.

6.1 Infected User Detection based on HTTP logs

When all other security measures fail, and malware gets into the network, early infected user detection and remediation is the last resort of defense. Since the successful progress of an attack requires some communication over the Internet (e.g. establishing a command and control channel), monitoring user activity in the network might reveal whether or not a user is infected. HTTP traffic log records are considered a sweet spot for the analysis because HTTP(S) is the most popular protocol that is usually allowed in all networks and through all firewalls, and hence frequently used by malware as well. Supervised learning then enables automated extraction of *behavioral patterns*, from the reference samples of malicious and legitimate network behavior, which generalize better than commonly used databases of malware signatures. For malware actors, it is far easier to change signatures like domain names, server IP addresses, or process hashes associated with malicious behaviors than the behaviors themselves, e.g., establishing command and control channels, sending spam, visiting advertisement servers, uploading private data, etc. Thanks to this generalization ability, data-driven models can detect zero-day attacks, polymorphic malware, file-less malware, and other advanced (signature-aware) threats designed to bypass traditional security solutions before significant damage can be done. On the other hand, supervised learning requires a significant amount of labeled data and a way of representing users' network behavior as numerical feature vectors, both of which are difficult to get. The labeled data, because labeling is an expert-requiring and labor-intensive process that can not be outsourced due to privacy issues, and the numerical representation, because it is

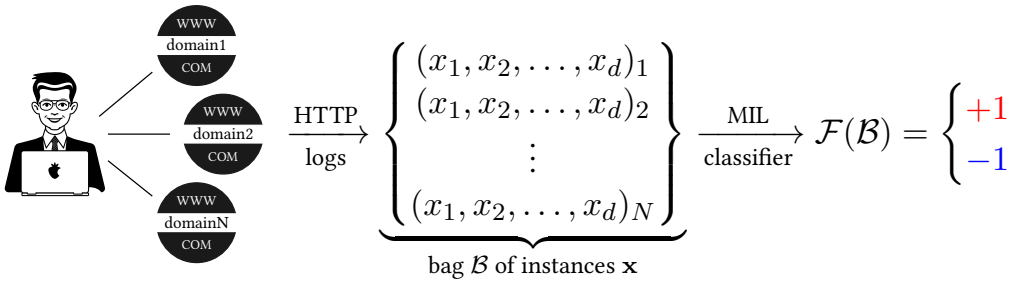

Figure 6.1: Classification of network users (as malware-infected or clean) formulated as a MIL problem. A user is modeled as a bag of instances (i.e. a set of feature vectors) that represent the user's communications with individual domains over HTTP protocol within a time window. Since a MIL classifier can consume bags of various sizes, it is not necessary to aggregate the instances into a single (possible lossy) user-level feature vector to account for the fact that each user can interact with a different number of domains. Moreover, embedding all communications into a single feature vector would complicate the subsequent identification of alarm-triggering communications during threat verification. An alternative per-communication approach $\mathcal{F}(\mathcal{B}) = \max_{\mathbf{x} \in \mathcal{B}} \mathrm{f}(\mathbf{x})$, using a traditional (single-instance) classifier $\mathrm{f}(\mathbf{x})$, would require a larger amount of (per-instance) labels and would be missing a global perspective on user behavior, resulting in limited detection capabilities.

unclear how users' network communications can be embedded into single feature vectors of a fixed dimension as each user may communicate with a different number of servers/domains[1].

We take advantage of the MIL paradigm (Section 2.1) to address both aforementioned problems. **Specifically, we define that MIL bags correspond to network users and bag instances to users' communications with (second-level) domains within a 24-hour time window** (Figure 6.1). Such formulation of the problem eliminates the need to represent every user with a single feature vector, as the flexibility of MIL in bag sizes reflects the reality that each user can establish a different number of communications within the time window. Next, the formulation reduces label acquisition costs because there is no need to pinpoint individual communications responsible for the infections, but it is enough to provide labels for whole users. Moreover, it opens new ways of acquiring labels. Since it is sufficient to know whether or not a user was infected in a particular time period, a completely separate source of data from network traffic data can be used for annotating (e.g. anti-virus reports) and thus benefit from cheaper and less ambiguous labels.

The proposed ISRT model (Chapter 4) further contributes to the above-mentioned MIL advantages by providing explanations of positive (user-level) predictions on the level of individual communications with domains. This capability is of great need, especially when the subsequent acting upon the raised alerts is associated with high costs (e.g. re-imaging users' computers), and the verdicts need to be well justified. Time spent on the justification can be significantly reduced if the top k recommended communications by the model include the relevant ones that are responsible for the infection.

To validate the applicability of the proposed MIL approach, we created a dataset by capturing HTTP log records from real network traffic of 100+ international corporate networks of various types and sizes that are customers of Cisco Global Threat Alerts. The objective is to classify

[1] A typical workaround is to create a model classifying users' communications rather than the users as a whole. However, individual communications may not carry enough discriminatory information to make verdicts about them. For example, a seemingly legitimate request to google.com might be in reality related to malicious activity when it is issued by malware checking the Internet connection. Similarly, visiting ad servers in low numbers is considered a legitimate behavior, but higher numbers might indicate Click-fraud infection. To make correct judgments in such cases, the broader context of user behavior must be considered.

Dataset	Training set	Validation set	Testing set
Date range	18-22 Jan 2021	3 Feb 2021	24 Feb 2021
HTTP logs	1,186,465,181	239,079,385	264,342,073
Communications	2,829,316	581,826	554,425
Infected users	1,830	430	380
Clean users	115,700	24,987	23,695

Table 6.1: Specification of the MIL dataset representing the problem of detecting malware-infected users in computer networks based on HTTP log records. Every user is represented as a bag of instances that correspond to the user's communications with individual domains. The numerical representation of each such communication is created by extracting URL features from respective HTTP log records. The counts in the table for HTTP logs and communications correspond to the sum over all users and illustrate the potentially higher labeling requirements on these lower levels.

users of these networks as either infected or clean. Each user is represented as a bag of instances, where the instances correspond to the individual user's communications with domains within 24 hours. The numerical representation of each such communication is computed from collected HTTP log records. The procedure is as follows. First, HTTP log records originating from a given user and targeting a particular second-level domain are grouped. Then, each log record is converted into a feature vector by extracting a set of prior-art features [79, 45, 76] from individual log fields. Specifically, we use a subset of URL-based features that are designed to differentiate between URL strings generated by malicious and legitimate applications. These are a few examples of the features: the number of occurrences of reserved URL characters (`"_","-","?","!","@","#","&","%"`), the digit ratio, the lower/upper case ratio, the vowel change ratio, the number of non-base64 characters, the maximum length of lower/upper case stream, the maximum length of consonant/vowel/digit stream, etc. In sum, there are 359 features. Finally, to represent the communication with a single instance, feature vectors of the grouped HTTP log records are aggregated using a maximum as the aggregation function. The dataset is divided into training, validation, and testing sets. The training set represents the period of five working days in January 2021. The validation and testing set then covers one working day of the first and the last Wednesday in February 2021, respectively. In total, there are 118,108 unique users (i.e. bags). On average, each user communicated with 24 domains (i.e. bag instances). More detailed statistics regarding the dataset are shown in Table 6.1.

We train the ISRT model (Chapter 4) with the following hyper-parameter settings: the number of trees to grow set to 100, the number of considered threshold values set to 8, and the number of epochs for training selectors set to 10. We also use similar hyper-parameter values (i.e. 100 trees and 8 threshold values) to train the BLRT model (Chapter 3). Furthermore, to train the MIL-NN model [91], we perform a grid search over the following configurations: the instance layer size $\{10, 30, 60\}$, the aggregation function type $\{$`mean`, `max`, `mean-max`$\}$, and the bag layer size $\{5, 10, 20\}$. We use rectified linear units (ReLU), ADAM optimizer, mini-batch of size 32, and the maximum number of epochs set to 1000. Finally, we train the MI-SVM classifier [7] for L2 regularization values $\lambda \in \{10^{-5}, 10^{-4}, \ldots, 1\}$ and the number of epochs set to 100. The final configuration for both prior-art models is selected based on the highest achieved performance on the validation set in terms of the area under the Precision-Recall curve.

Using the Precision-Recall and ROC curves, Figure 6.2 shows the efficacy results of the applied models on the validation and testing set separately. Different points on the Precision-Recall curve correspond to different decision threshold values of a particular model and indicate the percentage of alarms that are actually correct (precision) subject to the percentage of detected

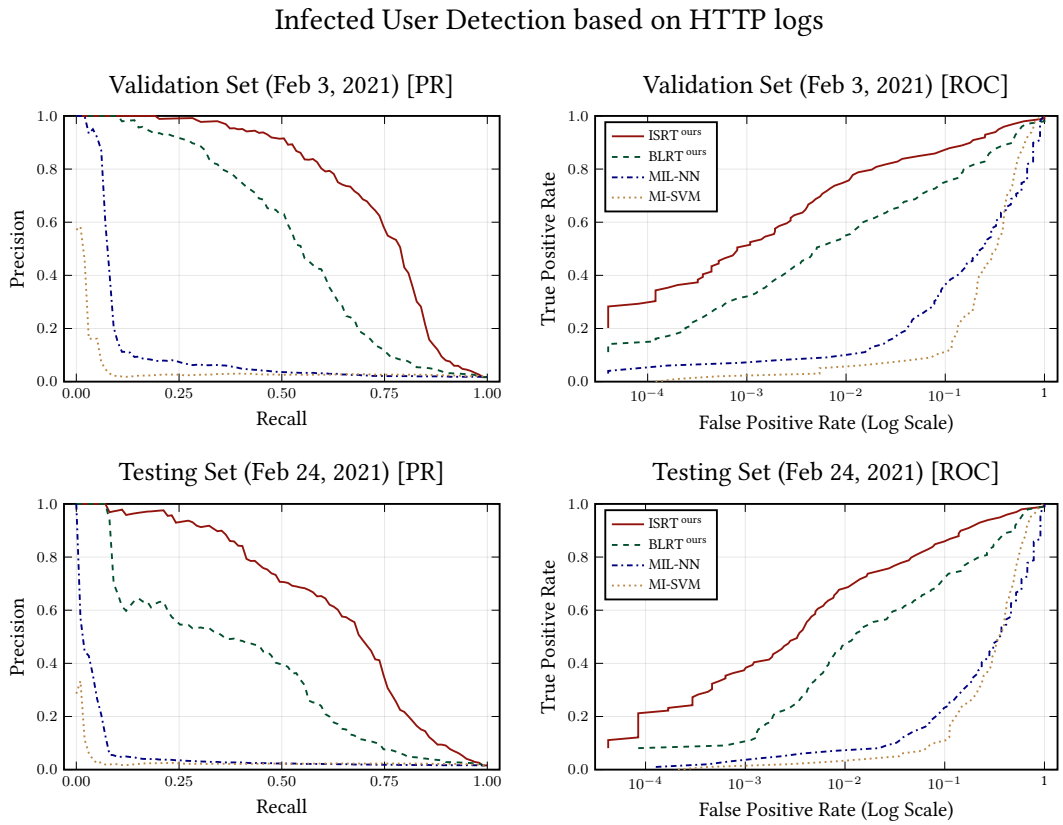

Figure 6.2: Precision-Recall and ROC curves of four MIL models evaluated on the validation (Feb 3, 2021) and testing (Feb 24, 2021) set of the infected user detection (HTTP) dataset.

infected users in the whole dataset (recall / true positive rate). Perfect recall (score 1) means that all infected users are detected, while perfect precision means that there are no false alarms. ROC curve then provides information about the volume of false alarms as the percentage of monitored healthy users (false positive rate)[2]. Points on the ROC curve might also serve for calculating precision under different imbalance ratios of infected to clean users [17].

As can be seen from Figure 6.2, the ISRT model has clearly superior performance as it produces equal or fewer false alarms than any other involved method at any arbitrary recall on both sets. The BLRT model ranks second best. It has a decent performance on the validation test (Feb 3, 2021), but on the testing set (Feb 24, 2021), the performance drops notably. This decrease in model performance over time is known as an aging effect. A model becomes obsolete as the distribution of incoming data shifts. Resistance to that is an important model characteristic, from an application point of view, albeit not always evaluated by researchers [86]. We attribute this decay to the fact that the BLRT model can extract only global bag-level univariate statistics computed across all instances within a bag. Because of this, it might be difficult to effectively separate a multivariate malicious signal hidden in a single instance from an abundant user background, which can evolve over time. On the other hand, the discriminative signal apparently does not lie on the local instance-level completely, as the instance-based classifier MI-SVM performs poorly. Probably the ability to combine these two approaches, by selecting and judging individual instances according to the need while collecting global evidence, might be the reason why the ISTR model excels in this task. Interestingly, the MIL-NN model is able to reliably detect

[2]While precision answers to the question: "With how big percentage of false alarms the network administrators will have to deal with?", false positive rate gives an answer to: "How big percentage of clean users will be bothered?".

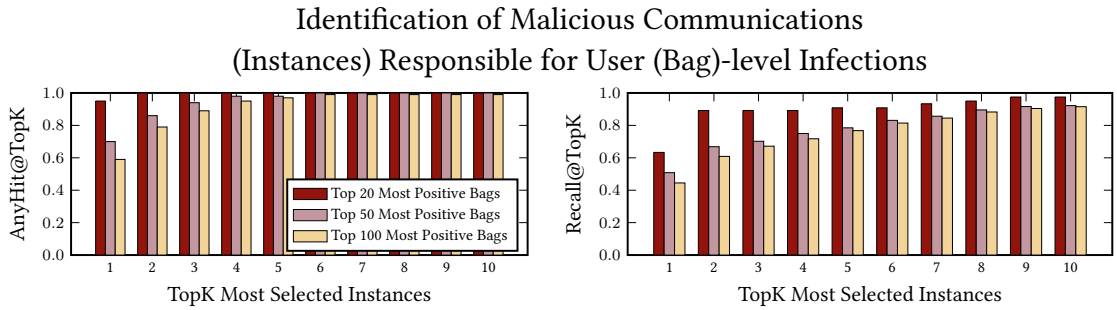

Figure 6.3: Assessment of ISRT's explanations as an information retrieval task. AnyHit@TopK (on the left) shows the percentage of bags for which at least one relevant (i.e. positively labeled) instance appeared among the TopK most selected instances. Since bags can contain multiple relevant instances, Recall@TopK (on the right) shows how many of them are among the TopK instances. The perfect Recall@1 can not be achieved unless all bags have only one relevant instance.

only a very limited number of infections, although, in the recent work [88], it has been shown to perform well on a similar task[3].

Quantitative assessment of ISRT's explanations: As mentioned in Section 4.4, the main benefit of the ISRT model over BLRT or the prior-art MIL-NN model, besides higher performance on some datasets, is the ability to explain the positive bag predictions. The provided explanations are in the form of assigned scores to individual instances (according to the number of times they have been selected during the bag class prediction; Algorithm 4), upon which they can be sorted and presented to the operator for judgment. Analogically to the information retrieval task, the goal is to place the most relevant instances at top positions. To assess this ability on the dataset, we used Cisco's internal deny list of known malicious domains to label instances[4] (i.e. communications with domains) of bags that have been classified as positive by the ISRT model on the testing set. For the first 20, 50, and 100 most positive bags (for which we had at least one positive instance-level label), we calculated AnyHit@TopK and Recall@TopK metrics. Figure 6.3 presents the results for the first top ten (TopK) most selected instances.

It can be observed from the left subplot of Figure 6.3 (AnyHit@TopK) that threat analysts investigating the first 20 most positive bags would encounter the first piece of evidence for the infection (i.e. any malicious domain) just by analyzing the top two recommended instances from each bag. This ability slightly decreases for the higher number of bags (i.e. 50 and 100), but it is still very useful considering the fact that the largest bags have over 100 instances, and on average, only two are labeled as malicious — a needle in a haystack problem. This can be also seen from the right subplot (Recall@TopK) showing that about 50% of all positive instances in bags can be discovered just by verifying the first (Top1) recommended instance.

The above results demonstrate that the ISRT model is capable of accurately classifying bags as well as promoting instances inside bags that are responsible for positive bag predictions.

[3]These results were published in our 2019 paper [64]. In light of what we know now, we believe that the utilization of standard *non-balanced* mini-batches throughout the learning is to blame for the poor performance of MIL-NN. When training neural networks on imbalanced data, *class-balanced* mini-batches may be of essential importance [103]. Unfortunately, due to data retention policy, we could not re-evaluate the MIL-NN model with class-balanced mini-batches on this dataset again.

[4]This way of identifying malicious communications is not so effective in production, since new threats are not on the deny list yet and need to be first discovered.

6.2 Infected User Detection based on Network Events

Modern intrusion detection systems (IDS) searching for ongoing malicious activities in computer networks typically use the ensemble strategy [48]. Rather than building a single, highly sophisticated detector, the problem is addressed independently by many relatively simple detectors whose findings (*network events*) are combined into a single final output. This allows to integrate (high-precision but low-recall) signature-based detectors and supervised classifiers with (low-precision but high-recall) unsupervised anomaly detectors and human-designed rule-based detectors, which, individually, would generate too many false alarms, but when combined, the ensemble outperforms any individual detector. This way, domain-expert knowledge can be encoded into a number of narrowly focused detectors that are simple to maintain and comprehend, without having to worry about their individual performance. Produced events then contain information about detected users' suspicious activities such as large data transfers, specific file-type downloads, initiated communications with raw IP addresses (unusual for humans), communications with algorithmically generated domain names (typical for DGA-based malware[5]), communication patterns that are too regular to be caused by a human, communications with domains/servers/countries that are unlikely for a given user based on its history or the history of its network peers, etc. The objective of an ensemble technique is to learn (in a supervised way) how to combine these network events to create a superior detector.

Several linear and non-linear combiners (classifiers) have been developed for this purpose, with a focus either on interpretability [8, 72] or accuracy at the top of returned samples [2, 82]. However, they all combine the events based on their hard/soft scores (by treating them as input features), ignoring any additional contextual information often present in the events, as shown in Figure 6.4. This extra information (e.g., about the number of uploaded/downloaded bytes in the case of `LargeDataTransfer` detector or a filename in the case of `ArchiveDownload` detector, etc.) is primarily addressed to analysts who may utilize it during threat alert investigations. **We aim to include this contextual information into the model decision process by framing the problem of combining network events as a classification of JSONs.** The hypothesis is that if the supporting information can be useful to threat analysts, then the model can benefit from including it as well.

To validate this, we created a dataset from real network traffic of 20 corporate networks. Samples in the dataset represent individual network users per 24-hour intervals and their corresponding network events as detected by the Cisco Global Threat Alerts IDS. Since the events are already JSON-formatted, no particular conversion is required. To make the problem non-trivial, events produced by the high-precision detectors are filtered out[6], and only 36 types of (weak signal) network events are kept. Each user is represented as a JSON object containing network events detected on that user within the 24-hour time window, as shown in Figure 6.4. If a user has multiple events of the same type, they are merged into a single one by concatenating their contextual JSON arrays (i.e. arrays named `SERVER_IP_ADDRESSES`, `URLS_AND_FILENAMES`, etc.). This step preserves all important information but helps to de-duplicate some repetitive JSON fields (i.e. `eventid`, `detectorid`, `severity`, etc.). The training part of the dataset spans the first week of August 2022 and contains 236,656 users, of which 1,906 are labeled as malware-infected. The testing part then covers the first week of the next month (September 2022) and contains 176,124 users, of which 1,853 are labeled as malware-infected.

First, we use the JSON2bag method (Chapter 5) to convert the dataset of JSONs into MIL bags, and then we train three MIL models: ISRT (Chapter 4), BLRT (Chapter 3), and prior-art MIL-NN [91]. Both ISRT and BLRT are trained using the default values of hyper-parameters and

[5]The Domain Generating Algorithm (DGA) is used by malware to periodically generate seemingly random domain names in an attempt to search for command and control servers that use a subset of these domains. The fact that the domains are not hard-coded in binaries and are constantly changing makes blocking of the malware difficult [93].

[6]The most reliable high-precision (signature-based) events are used as labels.

```
{
    "events": {
        "NELDATT1": {
            "detectorid": "LargeDataTransfer",
            "severity": 6,
            "keyvalueslong": {
                "UPLOADED_BYTES_COUNT": [389297230,150786381,1040993298],
                "DOWNLOADED_BYTES_COUNT": [4283298,185323,11008]
            },
            "keyvaluesstrings": {
                "SERVER_IP_ADDRESSES": [
                    "XXX.XXX.XXX.XXX",
                    /.../
                ]
            }
        },
        "NEARCFD1": {
            "detectorid": "ArchiveDownload",
            "severity": 5,
            "keyvaluestuples": {
                "URLS_AND_FILENAMES": [
                    {
                        "key": "http://download.com/file.gz",
                        "val": "file.gz"
                    },
                    /.../
                ]
            }
        },
        "NEAVGAS2": {
            "detectorid": "DetectionPersistentAutonomousSystem",
            "severity": 6,
            "keyvaluestuples": {
                "AUTONOMOUS_SYSTEMS_AND_SERVER_IP_ADDRESSES": [
                    {
                        "key": "AS24319 Akamai Technologies Tokyo ASN",
                        "val": "XXX.XXX.XXX.XXX"
                    },
                    /.../
                ]
            }
        },
        "NEAVGAC1": {
            "detectorid": "DetectionCountry",
            "severity": 5,
            "keyvaluestuples": {
                "COUNTRIES_AND_SERVER_IP_ADDRESSES": [
                    {
                        "key": "cc",
                        "val": "XXX.XXX.XXX.XXX"
                    },
                    /.../
                ]
            }
        },
        /... possibly other detected network events on a particular user /
    }
}
```

Figure 6.4: Example of network events detected on a user by Cisco Global Threat Alerts.

100 trees. Instance selectors of ISRT are optimized for 10 epochs. The architecture of MIL-NN is as follows: an input layer of size 119, the first hidden layer of size 200 followed by the `mean-max` aggregation, the second hidden layer of size 100, and two output neurons. The activation function is of type Rectified linear units (ReLU), and the weights are optimized using the AdaBelief optimizer for 10,000 training steps with randomly sampled class-balanced mini-batches of size

eight. Next, we convert the JSONs into hierarchical MIL bags using JsonGrinder [122] and train HMIL-NN [80] as a reflection of the JSON structure with the size of the inner dense layers set to 10, the aggregation function set to `mean-max`, and the activation function set to ReLU. Similarly, the network weights are optimized using the AdaBelief optimizer for 10,000 training steps with randomly sampled class-balanced mini-batches of size eight. Finally, we train two context-independent baseline models: Random Forest and Logistic Regression; by one-hot encoding the observed events, which are included in JSONs, into single feature vectors of size 36. Logistic Regression is trained with L2 regularization set to 10^{-4} and Random Forest with 100 trees.

Figure 6.5 summarizes results from the evaluation in terms of ROC curves for each of the six models. The horizontal x-axis has a logarithmic scale to emphasize the region of low False Positive Rate $[10^{-4}, 10^{-2}]$ that is relevant from the application point of view. Since 87% of users are in both the training and testing set, we not only do the standard time-based split evaluation (left subplot), but we also repeat it five times with various disjoint train/test user subsets (right subplot) to see if the models generalize (over users) and do not overfit by learning user-specific rather than malware-related characteristics. Ribbons around the ROC curves then show the standard deviations computed from these five rounds.

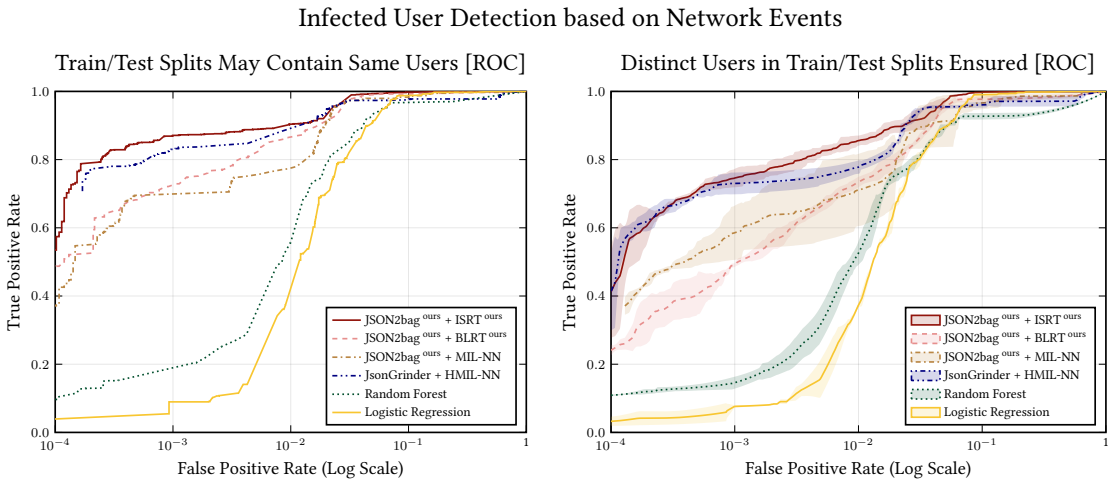

Figure 6.5: ROC curves of six models evaluated on the problem of detecting infected users in computer networks based on network events generated by Cisco Global Threat Alerts IDS. Individual network events describe various suspicious user activities, such as large data transfers, executable file downloads, unusual communication with servers, etc. A model's objective is to combine a multitude of these weak signals into a single, strong decision of whether a user is infected or not. While the baseline models (i.e. Logistic Regression and Random Forest) only work with the information about the types of events that have been detected on a particular user, the JSON-based models also take into account the contextual information attached to the events like filenames, domain names, URLs, transferred bytes, etc. Originally, this extra information was supplied only to human analysts (to facilitate incident investigation), as it was unclear how to encode it with a fixed number of features due to its variability in size, data type (numbers/strings), and structure. Since the events are already JSON-formatted, the ability to learn on JSONs overcomes this issue immediately without any need to develop a task-specific feature extraction component that would have to be updated as new types of events are added. To validate that the models do not overfit by learning user-specific rather than malware-related characteristics (as 87% of users are in both train/test sets), we not only do the standard time-based split evaluation (on the left), but we also repeat it five times with a different disjoint train/test user subsets (on the right).

In summary, Figure 6.5 indicates that by taking advantage of the contextual information, detection performance can be significantly improved. For instance, in the right subplot, at $\mathrm{FPR} = 10^{-3}$, the best-performing context-*independent* (baseline) model, Random Forest, has only $\mathrm{TPR} = 14.6\%$, while even the *worst*-performing context-*aware* (JSON-based) model, BLRT, achieves $\mathrm{TPR} = 48.1\%$, and the best one, ISRT, then $\mathrm{TPR} = 74.6\%$. Surprisingly, the ROC curve of the second-best model, HMIL-NN, is very similar to that of the ISRT model, especially on the FPR interval between 10^{-4} and 10^{-3}. As FPR approaches to 10^{-4}, HMIL-NN even marginally beats ISRT. When comparing the left and right subplots, it can be seen that the models tend to overfit the training users and perform slightly worse on new, unseen testing users. In particular, at $\mathrm{FPR} = 10^{-3}$, the detection rate (TPR) of ISRT drops by 12.5% point (from 87.1%) and of HMIL-NN by 10.1% point (from 83.1%). Nonetheless, the overfitting is not severe, and the ISRT and HMIL-NN models demonstrate that they can learn mostly relevant (non-user-specific) patterns from the JSON-formatted events and achieve detection rates up to 60% point higher than the baseline models.

6.3 Malicious Domain Discovery

Discovering new (previously unknown) malicious domains is essential for cyber defense, as only an up-to-date threat database can be successfully used for active threat blocking and/or model (re)training. Because of the high prevalence of legitimate domains, proposing randomly selected domains as candidates for discovering new malicious domains is highly inefficient. Instead, techniques leveraging existing knowledge about malicious domains are employed. An orthogonal approach to the already covered *behavior-based* modeling of network entities (e.g. users; Sections 6.1 and 6.2) is *relation-based* modeling. The idea is to build a graph of relations among network entities (e.g. users, domains, IP addresses, etc.) from the network data and make inferences about unknown properties (e.g. a probability of being malicious) of selected entities based on known properties of related entities. Such predictions are expected to be complementary to those based on the behavior modeling, as the respective feature spaces are independent of each other.

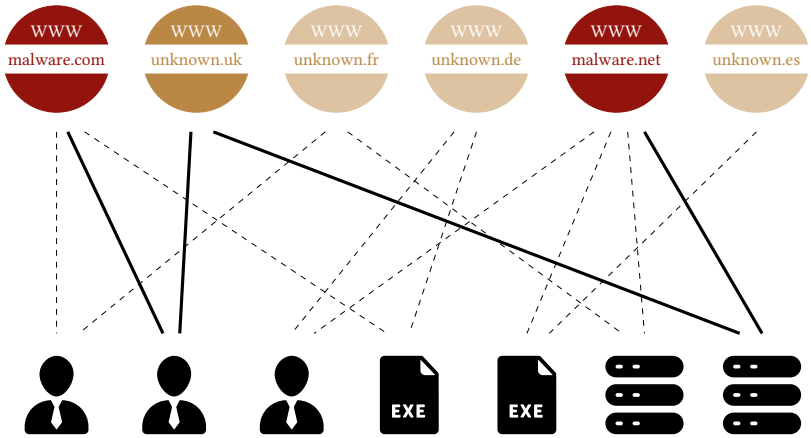

Figure 6.6: Heterogeneous graph representing interactions among four types of network entities: domains, users, files, and IPs; using three types of interactions: a user visited a domain, a domain was resolved to an IP address, and a binary file initiated communication with a domain. The graph is pruned to contain only known malicious domains (seeds) and unknown candidate domains that are related to at least one seed via a user/file/IP. The goal is to determine whether or not an unknown domain is high-risk malicious based on the local sub-graph formed by the node and its close (up to two edges away from the node) neighbors. A sub-graph induced by unknown.uk is highlighted with bold edges. Each entity may be described by its attributes.

The recent work [39] builds a heterogeneous graph (Figure 6.6) with four types of nodes (domain, IP address, user, and binary file) and three types of edges, each connecting a domain to one of the other (non-domain) entities. Specifically, a domain is connected to an IP/user/file if there is a record in the network data that the domain was resolved to the IP address, that the user visited the domain, or that the binary file initiated communication with the domain. Known malicious domains (seeds) are labeled in the graph using the current threat database, and the goal is to retrieve other (yet unknown) malicious domains. Furthermore, the task is refined by considering only domains two edges away from the seeds and eliminating domains that are most likely legitimate according to a proprietary (high-recall) algorithm. An output score of the algorithm (i.e. a probability of being malicious) is also used as one of the domain attributes. Additionally, the graph reduction step allows each remaining candidate domain to be further enriched with a list of VirusTotal's engines https://virustotal.com hitting on that domain. Such informative[7] but expensive enrichment would not be possible on the entire input data. The known malicious domains (seeds) are described by malware-related attributes such as the associated malware family and its risk rating. **The problem is formulated as a classification of unknown domains as high-risk malicious (or not) based on their sub-graphs extracted from the main graph as local neighborhoods surrounding the domains with a maximum path length of two.** Authors of the work [39] collected a dataset of 548 human-expert annotated domains (out of which 154 are the high-risk domains) between March 2021 and December 2021, and evaluated several variants of Graph Neural Networks (GNNs) [101] on the induced sub-graphs. Incidentally, the sub-graphs are stored as JSON objects (Figure 6.7), enabling us to approach the problem as a classification of JSONs.

To do this, we convert the JSONs into MIL bags using JSON2bag (Chapter 5) and train the ISRT model (Chapter 4) with 100 trees and ten epochs for the instance selector optimization. Next, we employ the BLRT model (Chapter 3) with default parameter values and 100 trees. Finally, we also apply the MIL-NN model [91] with the first hidden layer of size 200 followed by the `mean-max` aggregation, another hidden layer of size 100, and the two output neurons. The network uses rectified linear units (ReLU) as the activation function and its weights are optimized using the AdaBelief optimizer for 200 epochs with randomly sampled mini-batches of size ten. The hierarchical MIL approach represented by the tandem of JsonGrinder [122] and HMIL-NN [80] is not applied here because of schema incompatibilities among the JSON documents. Even after a moderate effort to resolve the schema incompatibilities, we could not make the method work. The best prior-art models trained by the authors of the dataset are Graph Neural Networks (GNNs) of type Residual Gated Graph Convolutional Neural Networks [20]. Evaluations are conducted using the graph as well as the node classification strategy. For more details about networks' architecture, hyper-parameters, and optimization, we refer the reader to the paper [39]. As baseline models, the authors of the paper use Random Forest and Logistic Regression to quantify the advantage of the graph-based models over the structure-agnostic ones. To form a single feature vector for each sub-graph representing a candidate domain, VirusTotal's engines are one-hot encoded and feature-based representations of nodes are summed over the node types and concatenated.

The prior-art models are evaluated in [39] using a five-times repeated random train/test split strategy, with 66% of the samples used for training and 34% for testing. To facilitate comparability, we follow the exact same evaluation protocol for ISRT, BLRT, and MIL-NN, including the identical list of domains used in each train/test split. The evaluation outcomes are presented in Table 6.2 using AUC, Accuracy, and Precision@20 metrics, and in Figure 6.8 as ROC curves. The displayed values correspond to the mean $\pm$ one standard deviation computed over the five

[7] Hitting VirusTotal's engines and the pre-processing algorithm score are weak indicators (not labels) in this task, because the goal is to recognize *high-risk* malware among a mix of lower-risk and legitimate domains. The presence of these indicators does not automatically imply a high threat severity. Investigation of high-risk domains is prioritized (over lower-risk ones) as not all suspicious domains can be manually analyzed due to the limited analyst resources.

```
{
    "domain": "unknown.com",
    "pos_probability": 0.7329,
    "vt_data": {
        "vt_labels": 1,
        "vt_hits": [
            "BitDefender:malware site",
            "Kaspersky:malware site",
            /... possibly other hitting VirusTotal's engines /
        ]
    },
    "context_data": {
        "ip": [
            {
                "name": "<ANONYMIZED_IP_ADDRESS>",
                "seeds": [
                    {
                        "domain": "sality-malware.com",
                        "clusterLabel": "sality malware",
                        "risk": 20,
                    },
                    /... possibly other known malicious domains (seeds) /
                ]
            },
            /... possibly other IP addresses to which unknown.com and at least one
            ↪ malicious domain resolve /
        ],
        "user": [
            {
                "name": "<ANONYMIZED_UNIQUE_USER_IDENTIFIER>",
                "seeds": [
                    {
                        "domain": "icedid-malware.com",
                        "clusterLabel": "icedid malware",
                        "risk": 15,
                    },
                    /... possibly other known malicious domains (seeds) /
                ]
            },
            /... possibly other users that visited unknown.com and at least one malicious
            ↪ domain /
        ],
        "file": [
            {
                "name": "<ANONYMIZED_FILE_SHA256_HASH>",
                "seeds": [
                    {
                        "domain": "lemonduck-malware.com",
                        "clusterLabel": "lemonduck malware",
                        "risk": 10,
                    },
                    /... possibly other known malicious domains (seeds) /
                ]
            },
            /... possibly other files associated with unknown.com and at least one
            ↪ malicious domain /
        ]
    }
}
```

Figure 6.7: Example of a sub-graph (unknown.com) stored in JSON format and containing relations to IPs, users, and files; and their relations to known malicious domains (seeds).

evaluation rounds. As can be seen from Figure 6.8, while the best-performing prior-art GNN model just marginally outperforms the reference (baseline) models, the JSON2bag + ISRT approach wins by a considerably larger margin. Not only does the JSON2bag + ISRT approach

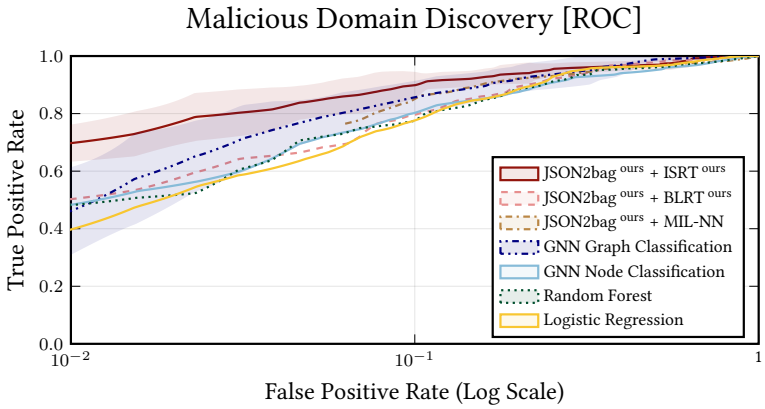

Figure 6.8: ROC curves of seven models classifying domains based on their sub-graphs of relations with network entities that are associated with known malicious activities. For the sake of clarity, a ribbon indicating the standard deviation around the mean value is shown only for our JSON2bag + ISRT technique and the best-performing prior-art method [39] denoted as GNN Graph Classification. The JSON2bag + ISRT technique outperforms the baseline structure-agnostic models (Random Forest and Logistic Regression) by a wider margin than the prior-art GNN Graph Classification method.

	AUC	Accuracy	Precision@20
JSON2bag ours + ISRT ours	**95.40** (1.92)	**91.49** (2.49)	**100.0** (0.00)
JSON2bag ours + BLRT ours	92.61 (1.90)	87.73 (2.29)	97.00 (2.74)
JSON2bag ours + MIL-NN	91.59 (1.72)	87.18 (1.93)	82.00 (13.04)
GNN Graph Classification	93.85 (2.22)	89.28 (1.59)	93.00 (6.71)
GNN Node Classification	90.82 (2.55)	87.51 (1.73)	92.00 (4.47)
Random Forest	91.84 (2.39)	87.07 (1.94)	95.00 (8.66)
Logistic Regression	91.72 (2.31)	86.30 (1.63)	94.00 (6.52)

Table 6.2: Numerical results of seven models evaluated on the problem of high-risk domain identification. From the application perspective, Precision@20 is the most relevant metric since it reflects analysts' daily/weekly sample validation limit. Apparently, the JSON2bag + ISRT method has the greatest potential for exploiting the network traffic graph structure.

have the dominant ROC curve, but also the highest mean values for all three metrics (Table 6.2). In particular, it is the only method that achieves the highest possible score of 100% in the Precision@20 metric. This metric, measuring the percentage of relevant samples among the top k (e.g. 20) proposed, is important from the application perspective since it reflects a daily/weekly limit on the number of samples that analysts can validate. Recall that threat analysis is a costly and time-consuming process, so only a tiny fraction of all unknown domains seen in the network data every day/week can be investigated. To maximize resource utilization, only relevant high-risk domains should be among the top k proposed ones. Interestingly, the next two models with the highest Precision@20 metric are tree-based as well: JSON2bag + BLRT ($97\% \pm 2.74\%$) and Random Forest ($95\% \pm 8.66\%$). This demonstrates that the herein proposed BLRT (Chapter 3), ISRT (Chapter 4), and JSON2bag (Chapter 5) tools can also be successfully used for modeling network graph structures.

6.4 Malicious Email Detection

Malicious emails are currently the major security threat to individuals and organizations [41]. This is mainly due to the ease with which they can be distributed, even by inexperienced cybercriminals, compared to remote attacks. Threat actors typically craft emails using social engineering tactics to convince a victim to visit a malicious URL, pay a false invoice, reveal credentials or other sensitive information, infect its device, or spread malware executable. One of the email threat defense systems utilized by Cisco in their email security solution http://cisco.com/go/emailsecurity is Cognitive Anti-Phishing Engine [106]. It is an ensemble of 30+ heterogeneous detectors that examine the header, content, attachments, sender reputation, and sender-recipient relationship of a given email to determine whether the email should be classified as phishing. The ensemble contains both types of detectors, the *rule-based* detectors like the link masquerade detector recognizing links such as `<a href="good.com">evil.com</a>` where the displayed domain differs from the actual one or the cryptocurrency address detector, and the *data-driven* detectors, which use, for instance, language models [34] to recognize the urgency and call-to-action in sentences[8]. For a given input email, the anti-phishing engine outputs a JSON object containing an array of triggered detections on that email, as shown in Figure 6.9. Apart from the score and the detector code name, each detection contains metadata with additional information related to the detection such as the particular sentence with the urgency, the detected cryptocurrency address, including the coin type (e.g. Bitcoin), or the displayed and the actual domain in the case of the link masquerade detector. Analysts subsequently use this metadata as evidence when they are justifying engine verdicts.

In the current implementation, the binary verdicts are made by combining detection scores linearly and comparing the results to a threshold value. **We aim to challenge this by training a model that operates on the entire JSON outputs, considering not only the detection scores but also the supporting metadata.** To this end, we use two proprietary datasets of emails collected during March and April 2022, respectively. Each dataset contains 50,000 positive and 50,000 negative emails. Positive emails represent a mixture of spam, phishing, and malware messages. Negative emails, on the other hand, are proven to be harmless messages (i.e. conversations, notifications, marketing) based on a manual analysis or prior knowledge. More specifically, the negative emails represent *hard examples* from the classification perspective as they were initially misclassified as positives by an alternative email security engine and later resolved as false positives by an analyst or a maintained set of rules. Because of this, evaluation results may be more pessimistic than they would be if the negative emails followed their natural distribution.

After processing both email datasets (March and April) by the Cognitive Anti-Phishing Engine, the following six models are evaluated upon the output JSONs. First, we transform the JSONs into MIL bags using JSON2bag (Chapter 5) and train the ISRT model (Chapter 4) with 100 trees and ten epochs for optimizing instance selectors, the BLRT model (Chapter 3) with 100 trees, and the MIL-NN model [91] with the first hidden layer of size 200 followed by the `mean-max` aggregation, two hidden layers of size 200 and 100, and the two output neurons. Then, we also transform the JSONs into *hierarchical* MIL bags using JsonGrinder [122] and train the HMIL-NN model [80] with the size of the inner dense layers set to ten and the aggregation function set to `mean-max`. Both neural network-based models use rectified linear units (ReLU) as the activation function and are optimized using AdaBelief optimizer for 1,000 training steps with randomly sampled mini-batches of size ten. Ultimately, we transform the JSONs into single feature vectors where features correspond to individual detector scores and train Random Forest with 100 trees and Logistic Regression with L2 regularization set to 10^{-4}. The Logistic Regression model simulates the currently used linear combiner in the engine.

[8] These are a few examples of urgent and call to action sentences: "*Final warning about your unsuccessful tax payment.*", "*Click here immediately to reset your password.*" or "*Download and save a copy of your payment.*".

```
{
    "detections": [
        {
            / Link Masquerade Detector /
            "code": "DLINKMASQ",
            "score": 1.0,
            "meta": [
                {
                    "id": "displayed",
                    "val": "good.com"
                },
                {
                    "id": "real",
                    "val": "evil.com"
                }
            ]
        },
        {
            / Cryptocurrency Address Detector /
            "code": "DCRYPTOCCY",
            "score": 1.0,
            "meta": [
                {
                    "id": "address",
                    "val": "13AD9cJjGm1UaNRfLFsw66V4J5RC7fYQx5"
                },
                {
                    "id": "cointype",
                    "val": "bitcoin"
                }
            ]
        },
        {
            / Call-To-Action Detector /
            "code": "DCTA_OTH",
            "score": 1.0,
            "meta": [
                {
                    "id": "segment",
                    "val": "Send 1 BTC to this address:
                    ↪  13AD9cJjGm1UaNRfLFsw66V4J5RC7fYQx5."
                }
            ]
        }
    ]
}
```

Figure 6.9: Example of Cognitive Anti-Phishing Engine's output for a malicious email.

All six models are evaluated two times, first by using the 5-times repeated 3-fold *cross-validation* strategy on the March dataset and then using the *time-based* split, where the older data from March is used for training and the more recent data from April is used for testing. Figure 6.10 shows the ROC curves of each model for both evaluation strategies separately. In the case of the cross-validation strategy, the curves correspond to the mean value, and the ribbon around the curves to one standard deviation computed over the folds. **As seen from the figure, the efficacy results can be considerably improved by using the metadata along with the detection scores.** In both evaluations, the dominant ROC curve belongs to the JSON2bag + ISRT technique, particularly in the application-relevant interval of low FPR $[10^{-4}, 10^{-3}]$. For instance, on the time-based split evaluation at $\mathrm{FPR} = 10^{-3}$, the ISRT model correctly detects 42.5% of all positive emails ($\mathrm{TPR} = 0.425$) while the Logistic Regression model only 9.7% ($\mathrm{TPR} = 0.097$). When comparing the results from the cross-validation and the time-based split evaluation, it can be seen that the results from the cross-validation are more optimistic (i.e. higher values of TPR for the same FPR). This is because, in cross-validation, future samples are accessible in the

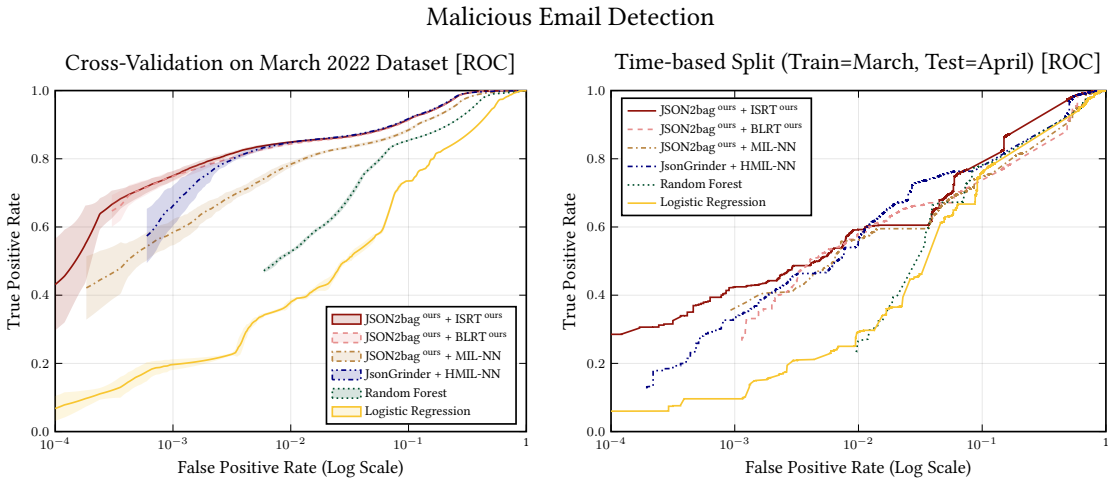

Figure 6.10: ROC curves for six models predicting whether an email is malicious or not based on a list of detections provided by Cognitive Anti-Phishing Engine. Unlike the linear (Logistic Regression) and the non-linear (Random Forest) detection score combiner, the JSON-based models also consider the contextual information surrounding the detections that was originally intended only for human analysts justifying the final verdicts. The ISRT model appears to benefit most from this extra information, especially in the range of low FPR $[10^{-4}, 10^{-3}]$ where it outperforms all other methods in the percentage of correctly detected malicious emails (TPR). The difference between the cross-validation (on the left) and the time-based split (on the right) demonstrates the necessity to respect the arrival time of email data samples during evaluation, as not doing so (cross-validation) may lead to overly optimistic results that might not be delivered once a model is deployed in production.

training set due to random shuffling during fold formation, which is problematic for data that evolves over time (i.e. concept-drift [57] is present in the data), such as the email data. Therefore, cross-validation values of FPR and TPR cannot be expected to be matched in production since a model cannot peek forward in time on a real-time stream of email data. Moreover, the cross-validation results also falsely indicate that the non-linear Random Forest model is a better score combiner than the currently employed linear model. And that the interpretability of the linear model may be exchanged for the improved detectability of the nonlinear model, which would be a poor trade-off, as the time-based evaluation demonstrates. This illustrates the importance of the time-based split evaluation on data with concept-drift as the cross-validation strategy does not reveal whether a classifier learns patterns that do not generalize over time. This is especially true for complex learners, such as the JSON-based models, which have a greater chance of learning patterns that do not apply in new situations due to the richer data available to them.

6.5 Malicious Executables Classification

Classifying files that are executable on end-user devices is at the core of anti-malware protection. This task is represented by a publicly available Avast-CTU CAPE Dataset [16]. The dataset comprises 48,976 malicious samples collected between the years 2017 and 2019, categorized into ten malware families. Each sample is defined as a JSON report produced by CAPEv2 analyzer [84] monitoring the sample's execution in a sandbox environment. The JSON report, therefore, contains not only the commonly used *static properties* that can be extracted without running the file, such as the metadata from PE headers [6, 52, 97], a list of imported libraries, sections, etc., but also the results from the *behavioral analysis* regarding accessed files, registry keys, mutexes, API calls, etc., collected during the file execution. Such behavior data is more expensive to obtain,

	Malware classification based on CAPEv2 sandbox JSON reports							
	Behaviour & Static features				Static features only			
	JsonGrinder	JSON2bag ours			JsonGrinder	JSON2bag ours		
	HMIL-NN	MIL-NN	BLRT ours	ISRT ours	HMIL-NN	MIL-NN	BLRT ours	ISRT ours
Adload	**100.00**	**100.00**	**100.00**	**100.00**	60.00	80.00	**100.00**	**100.00**
Emotet	86.54	88.30	86.91	**92.07**	2.35	2.16	13.39	**25.44**
HarHar	**98.55**	**98.55**	**98.55**	**98.55**	**98.55**	**98.55**	95.65	**98.55**
Lokibot	95.33	**96.20**	87.74	95.91	**90.07**	79.85	76.50	80.15
Qakbot	99.24	99.24	93.64	**99.49**	35.62	**50.78**	28.21	29.78
Swisyn	99.88	99.62	99.39	**99.95**	99.72	99.39	99.91	**99.95**
Trickbot	**98.58**	96.18	89.37	97.75	68.49	**87.80**	85.25	85.25
Ursnif	94.02	94.02	53.39	**98.61**	70.32	**85.66**	29.88	47.61
Zeus	**95.83**	93.75	81.94	94.44	**82.64**	70.92	74.47	81.56
njRAT	**99.26**	92.27	87.66	98.53	75.51	74.49	85.21	**86.32**
Overall	94.5	94.84	90.69	**96.81**	63.0	63.21	63.84	**68.79**

Table 6.3: Per-class and overall accuracy of four methods classifying malware samples into ten families based on CAPEv2 sandbox JSON reports. The methods are evaluated using either behavior and static analysis data or solely data from the less expensive static analysis. In both cases, JSON2bag combined with the ISRT model yields the best overall results. Results for the pair JsonGrinder and HMIL-NN are taken from the prior-art work [16]. A dummy model always predicting the dominant class (Emotet) would have 30.78% accuracy.

but it's also more complicated for attackers to manipulate in order to evade detection, unlike static data, which can be disordered using the techniques of obfuscation[9] and/or encryption[10]. An example of a CAPE's JSON report, including both the `static` and the `behavior` features, can be seen in Figure 6.11. Authors of the dataset advise to make the train-test split based on a certain date to account for the concept drift present in the malware data. To train a model (JsonGrinder + HMIL-NN), they consider (and we follow it) all samples up to 2019/08/01 to be a part of the training set (37,512 samples) and all samples after that to be a part of the testing set (11,464 samples).

The prior-art model [84], JsonGrinder [122] + HMIL-NN [80], is constructed with the size of the inner dense layers set to 32, the aggregation function set to `mean-max`, and the activation function set to ReLU. The network weights are optimized using ADAM optimizer with randomly sampled mini-batches of size 500 for 200 training steps. In addition to the prior-art model, we convert the JSON reports to MIL bags using the JSON2bag tool (Chapter 5) and train three additional MIL models upon the converted JSONs: MIL-NN [91], BLRT (Chapter 3) and ISRT (Chapter 4). The MIL-NN architecture starts with the input layer of size 119 that is fully connected to the first hidden layer of size 300 followed by `mean-max` aggregation. It continues with another two dense layers of size 300 and 200 and ends with the output (class) layer of size 10. All activation functions are of type rectified linear unit (ReLU). The network weights are optimized via AdaBelief for 200 learning steps with randomly sampled class-balanced mini-batches of size 200 (i.e. 20 samples per class). BLRT and ISRT models are trained with default parameter settings and one-vs-rest multi-class strategy, where an ensemble of 32 trees per class is trained.

Table 6.3 summarizes results from the evaluation of all four models on the testing set in terms of per-class and overall accuracy. The evaluation is made separately for the combination of

[9] Obfuscation renders the code more difficult to read and understand without altering the essence of the code. Typically, it replaces code segments with randomized variants that have the same effect, modifies all the names, reorders independent parallel computations, inserts extra code not affecting the payload, etc.

[10] An encrypted executable contains a simple bootstrap code that decodes the rest of the program and executes it. Since the techniques of the code encryption and obfuscation are also sometimes used to conceal proprietary algorithms, their presence alone does not automatically imply malicious intent.

the behavior and static features and solely for the static features, which are less expensive as they can be extracted without running the file. In both scenarios, JSON2bag in tandem with the ISRT model achieves the best overall accuracy. Specifically, the level of accuracy is 96.81% on the full JSON reports and 68.79% on the reduced reports containing only the static data. Interestingly, the overall accuracies provided by the two neural network-based models are quite close to one another, about 94.5% and 63% on the full and the reduced JSON reports, respectively. This raises the question of whether it is necessary to model the problem using the more complex *hierarchical* MIL approach when the *non*-hierarchical one offers similar performance. The BLRT model achieves the overall accuracy of 90.69% on the full reports and 63.84% on the reduced ones, and as the only model, it is not superior in any class. A dummy model constantly predicting the most frequent class (Emotet) has 30.78% accuracy in both scenarios. To sum up, the behavior data significantly improves models' classification performance (using the static data solely, only three malware families, Adload, HarHar, and Swisyn, are detected with the same level of accuracy as when the static and behavior data are combined), and the best technique for this problem proved to be JSON2bag paired with ISRT.

```
{
   "behavior": {
      "summary": {
         "keys": [
            "DisableUserModeCallbackFilter",
            "HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Windows NT\\CurrentVersion\\GRE_Initialize",
            "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\GRE_Initialize\\DisableMetaFiles"
         ],
         "resolved_apis": [
            "kernel32.dll.SetEndOfFile",
            "kernel32.dll.GetProcessHeap",
            /.../
         ],
         "executed_commands": [],
         "write_keys": [],
         "files": [
            "C:\\Windows\\System32\\api-ms-win-core-fibers-l1-1-1.DLL",
            "C:\\Windows\\System32\\api-ms-win-core-localization-l1-2-1.DLL",
            /.../
         ],
         "read_files": [
            "C:\\Users\\comp\\AppData\\Local\\Temp\\config-nkxmr.json"
         ],
         "started_services": [],
         "created_services": [],
         "write_files": [],
         "delete_keys": [],
         "read_keys": [
            "DisableUserModeCallbackFilter",
            "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\GRE_Initialize\\DisableMetaFiles"
         ],
         "delete_files": [],
         "mutexes": []
      }
   },
   "static": {
      "pe": {
         "icon_hash": "0304a765b7d1981a90d07ce1b4f8148c",
         "sections": [
            {
               "raw_address": "0x00000400",
               "name": "UPX0",
               "characteristics": "IMAGE_SCN_CNT_UNINITIALIZED_DATA|IMAGE_SCN_MEM_EXECUTE|IMAGE_SCN_MEM_READ|IMAGE_SCN_MEM_WRITE",
               "virtual_size": "0x000e8000",
               "virtual_address": "0x00001000",
               "size_of_data": "0x00000000",
               "entropy": "0.00",
               "characteristics_raw": "0xe0000080"
            },
            /.../
         ],
        /.../
         "guest_signers": {
            "aux_error": true,
            "aux_valid": false,
            /.../
            "aux_error_desc": "No signature found. SignTool Error File not valid
             ↪  C\\Users\\comp\\AppData\\Local\\Temp\\00282BE7FA944B73A114.exe"
         },
         "actual_checksum": "0x0005ce31",
         "imports": [
            {
               "dll": "ADVAPI32.dll",
               "imports": [
                  {
                     "name": "LsaClose",
                     "address": "0x541fb8"
                  }
               ]
            },
            /.../
         ],
         "exported_dll_name": null,
         "dirents": [
            {
               "name": "IMAGE_DIRECTORY_ENTRY_EXPORT",
               "virtual_address": "0x00000000",
               "size": "0x00000000"
            },
            /.../
         ],
         "versioninfo": [],
         "resources": [
            {
               "name": "RT_BITMAP",
               "filetype": null,
               "offset": "0x0012b860",
               "language": "LANG_CHINESE",
               "sublanguage": "SUBLANG_CHINESE_SIMPLIFIED",
               "size": "0x00000c68",
               "entropy": "7.74"
            }
            /.../
         ],
         "pdbpath": null,
         "osversion": "5.1",
         "icon_fuzzy": "00f093ca5233c12c497c0f3fc1557273",
         "imagebase": "0x00400000",
         "imported_dll_count": 4,
         "timestamp": "2019-03-26 02:18:56"
      }
   }
}
```


Figure 6.11: Example of the CAPE sandbox analysis output for a malicious executable.

Chapter 7

Conclusion

This thesis deals with end-to-end model learning from JSON-formatted tree-structured data. Since JSONs can capture information of variable size, structure, and data type, they offer far more flexibility for describing objects of interest than traditional fixed-size vectors of numbers. However, their non-vectorial nature complicates the subsequent use of ML methods. To avoid tedious (and possibly lossy) feature engineering, we propose a JSON2bag tool converting arbitrary JSON samples into MIL bags (i.e. sets of feature vectors) and two *general* MIL classifiers, BLRT and ISRT, capable of learning from any datasets of labeled bags.

JSON2bag (Chapter 5) is a feature extraction algorithm aiming to numerically represent all the information contained in JSON values (strings, numbers, booleans, and nulls) and the JSON tree topology (nested composition of objects and arrays). It flattens an input JSON sample into a one-level set of path-value pairs and extracts a pre-defined set of features from each such pair to create a set of feature vectors representing that sample. This multi-feature vector representation is compatible with MIL algorithms and can be created with a minimum (to none) information loss, in contrast to attempts to fit every JSON sample into a single fixed-length feature vector. Moreover, we demonstrate (Section 5.2) that, unlike prior-art techniques, JSON2bag can handle signal displacements and be applied even to schema-less problems (e.g. classification of HTML source codes). We also show (Table 5.5) that ISRT combined with JSON2bag can explain decisions by means of the top most important path-value pairs, which may facilitate the validation of predictions and provide insights into complex JSON data structures.

BLRT (Chapter 3) extends traditional ensembles of decision trees to the MIL setting. To account for the fact that bags may contain multiple feature vectors (i.e. instances), we adjust the standard node-splitting condition — testing whether a feature value is greater than a certain threshold — by an additional parameter judging the percent of instances that accomplish that condition. This can be interpreted as a quantile-based bag embedding, in which the embedding and node-splitting parameters are optimized jointly during the tree construction process. Since BLRT decides based on the global bag-level statistics, unlike most prior-art tree-based MIL methods, it can handle problems where the discriminative information is distributed over more than one instance. On 29 benchmark MIL datasets, we demonstrate (Section 3.2) that, despite its conceptual simplicity, BLRT significantly outperforms any of the prior 28 MIL classifiers, even without hyper-parameter tuning.

ISRT (Chapter 4) is another tree-based algorithm proposed in this thesis for solving MIL classification problems. Unlike BLRT, considering all bag instances in every tree node, ISRT selects a single instance at each step and makes the standard feature threshold decision upon it. The selection mechanism is implemented via an additional vector of parameters optimized in each tree node individually. ISRT is therefore able to extract patterns from both ends of the spectrum, the local instance-level (by selecting the same instance multiple times) and the global

bag-level (by selecting different instances). Learning to select only one (relevant) instance at each step also has a regularization effect and prevents noisy instances from affecting the decision-making process. Moreover, we observe (Table 4.2) that only a small fraction of bag instances is usually selected, and the rest is untouched. This makes it easier for analysts to find evidence for positive bag predictions on the level of instances, as they can skip over the untouched instances and focus on the selected ones only. Finally, we qualitatively (Table 5.5) and quantitatively (Figure 6.3) validate that ranking instances according to the number of times they are selected during the prediction reflects their importance.

Applications (Chapter 6) The above-presented tools provide a fully automated approach for deriving classification models from labeled JSON data samples. Since there is no need for task-specific feature extraction, the approach is universally and immediately applicable to a wide variety of problems, especially when the data are already in the JSON format or can be easily converted to it. In Chapter 6, we demonstrate this on five real-world cybersecurity problems, where the JSON representations of modeled objects, such as network users, domains, emails, and executable files, are already existing in the studied security systems (the JSON format is typically used for its machine and human readability, data type variability, structure mutability, and size flexibility), yet much of the information contained inside these JSONs is ignored by the deployed ML models. We show that our JSON2bag with BLRT/ISRT models can significantly improve the detection accuracy of studied systems by learning from all the information available in the JSON representations instead of just a few manually extracted values. To make sure that the observed improvements are not merely the products of overfitting, which could be a natural consequence of expanding the feature space, we apply strict evaluation protocols that include utilizing appropriate metrics, respecting the time-dependence in train/test dataset splits, and ensuring that the train/test parts contain distinct entities. Interestingly, when comparing BLRT vs. ISRT on these five real-world cybersecurity datasets, ISRT turns out to be superior, although, on several public MIL datasets (Table 4.1), the opposite was true. This can be explained by the difficulty of cybersecurity datasets, which are plagued by concept drift and various forms of noise, leading to a higher risk of overfitting with the BLRT model taking into account every instance in each decision node. ISRT's ability to resist overfitting (even in small data regimes) is due to its instance selection mechanism, which can be viewed as an extension of the feature selection mechanism used in the traditional decision trees to the dimension of instances.

To conclude, we firmly believe that the proposed techniques can unlock value in many other JSON/MIL datasets (even outside the cybersecurity domain) that conventional ML models have not effectively utilized yet. We also hope that the flexibility of JSON-based representations, coupled with the straightforward application of the proposed tools, has the potential to change the way how some ML problems are approached. Instead of spending time building and maintaining feature extraction pipelines, the focus could be put on creating richer JSON representations (perhaps with more bits of important information) from which models can be automatically derived with the proposed tools, resulting in more accurate and effective outcomes.

In future work, we would like to introduce a gradient-boosted version of ISRT. Ideally, this should produce even better-performing models as the optimization of the feature threshold and instance selector parameters would be governed in the whole ensemble by a single global loss function via residual values, similarly as did in popular XGBoost [25] or LightGBM [58] algorithms. Another interesting direction of research would be an attempt to interpolate between BLRT and ISRT by making the split node decision upon an aggregate value (e.g. mean) computed from a subset of selected bag instances. The subset size could vary among nodes, from one instance (ISRT) to all instances (BLRT), and correspond e.g. to JSON values of one specific JSON array/object. Finally, and perhaps most importantly, we plan to increase the visibility of the developed tools among researchers and ML practitioners by promoting the idea of learning on JSONs and creating easy-to-use libraries with clear documentation and examples.

Bibliography

[1] F. Abuzaid, J. K. Bradley, F. T. Liang, A. Feng, L. Yang, M. Zaharia, and A. S. Talwalkar. Yggdrasil: An optimized system for training deep decision trees at scale. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[2] L. Adam, V. Mácha, V. Šmídl, and T. Pevný. General framework for binary classification on top samples. *Optimization Methods and Software*, 37(5):1636–1667, 2022.

[3] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 52(6):1–28, 2019.

[4] M. Allen and D. Cervo. *Multi-Domain Master Data Management: Advanced MDM and Data Governance in Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2015.

[5] J. Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artif. Intell.*, 201:81–105, aug 2013.

[6] H. S. Anderson and P. Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, abs/1804.04637, 2018.

[7] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

[8] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78, 2018.

[9] B. Babenko. Multiple instance learning: algorithms and applications. *View Article PubMed/NCBI Google Scholar*, pages 1–19, 2008.

[10] B. Babenko, S. Belongie, and M. H. Yang. Visual tracking with online multiple instance learning. *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pages 983–990, 2009.

[11] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc., 2020.

[12] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[13] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, volume 3951, pages 404–417, 07 2006.

[14] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 2012.

[15] H. Blockeel, D. Page, and A. Srinivasan. Multi-instance tree learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 57–64, New York, NY, USA, 2005. Association for Computing Machinery.

[16] B. Bosansky, D. Kouba, O. Manhal, T. Sick, V. Lisy, J. Kroustek, and P. Somol. Avast-ctu public cape dataset, 2022.

[17] J. Brabec, T. Komárek, V. Franc, and L. Machlica. On model evaluation under non-constant class imbalance. In *Computational Science – ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV*, page 74–87, Berlin, Heidelberg, 2020. Springer-Verlag.

[18] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, aug 1996.

[19] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. CRC press, 1984.

[20] X. Bresson and T. Laurent. Residual gated graph convnets. *CoRR*, abs/1711.07553, 2017.

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[22] R. C. Bunescu and R. J. Mooney. Multiple instance learning for sparse positive bags. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 105–112, New York, NY, USA, 2007. Association for Computing Machinery.

[23] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon. Multiple instance learning: A survey of problem characteristics and applications. *Pattern Recognition*, 77:329–353, may 2018.

[24] S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1:300–307, 2007.

[25] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[26] Y. Chen, J. Bi, and J. Z. Wang. Miles: Multiple-instance learning via embedded instance selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12):1931–1947, dec 2006.

[27] V. Cheplygina, D. M. Tax, and M. Loog. Multiple instance learning with bag dissimilarities. *Pattern Recognition*, 48(1):264–275, 2015.

[28] V. Cheplygina and D. M. J. Tax. Characterizing multiple instance datasets. In *Similarity-Based Pattern Recognition*, pages 15–27. Springer International Publishing, 2015.

[29] Y. Chevaleyre and J. Zucker. Solving multiple-instance and multiple-part learning problems with decision trees and rule sets. application to the mutagenesis problem. In *Canadian Conference on AI*, volume 2056 of *Lecture Notes in Computer Science*, pages 204–214. Springer, 2001.

[30] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[31] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[32] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, dec 2006.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, jun 2019. Association for Computational Linguistics.

[35] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell.*, 89(1–2):31–71, jan 1997.

[36] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1422–1430, 2015.

[37] P. M. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55:78 – 87, 2012.

[38] L. Dong. *A comparison of multi-instance learning algorithms*. PhD thesis, The University of Waikato, 2006.

[39] S. Dvorak, P. Prochazka, and L. Bajer. Gnn-based malicious network entities identification in large-scale network data. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, page 1–4. IEEE Press, 2022.

[40] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image, 2016.

[41] Federal Bureau of Investigation. Internet crime report. https://www.ic3.gov/Media/PDF/AnnualReport/2021_IC3Report.pdf, 2021. Accessed: 2023-01-16.

[42] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(90):3133–3181, 2014.

[43] F. Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine learning research*, 5(9), 2004.

[44] J. R. Foulds. Learning instance weights in multi-instance learning. University of Waikato, Department of Computer Science, 2008.

[45] V. Franc, M. Sofka, and K. Bartos. Learning detector of malicious network traffic from weak labels. In *Machine Learning and Knowledge Discovery in Databases*, pages 85–99, Cham, 2015. Springer International Publishing.

[46] E. Frank and X. Xu. Applying propositional learning algorithms to multi-instance data. 2003.

[47] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 04 2006.

[48] M. Grill. *Combining network anomaly detectors*. PhD thesis, Czech Technical University, 2016.

[49] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. *Proceedings of the Nineteenth International Conference on Machine Learning*, page 179–186, 2002.

[50] R. Haluška, J. Brabec, and T. Komárek. Benchmark of data preprocessing methods for imbalanced classification. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2970–2979, 2022.

[51] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[52] R. Harang and E. M. Rudd. SOREL-20M: a large scale benchmark dataset for malicious PE detection, 2020.

[53] Herrera, Francisco and Ventura, Sebastián and Bello, Rafael and Cornelis, Chris and Zafra, Amelia and Sánchez-Tarragó, Dánel and Vluymans, Sarah. *Multiple instance learning : foundations and algorithms*. Springer, 2016.

[54] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[55] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[56] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[57] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC, Aug. 2017. USENIX Association.

[58] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[59] E. D. Knapp and J. T. Langill. Chapter 11 - exception, anomaly, and threat detection. In E. D. Knapp and J. T. Langill, editors, *Industrial Network Security (Second Edition)*, pages 323–350. Syngress, Boston, second edition edition, 2015.

[60] J. Kohout, T. Komárek, P. Čech, J. Bodnár, and J. Lokoč. Learning communication patterns for malware discovery in https data. *Expert Systems with Applications*, 101:129–142, 2018.

[61] J. Kohout, C. Škarda, K. Shcherbin, M. Kopp, and J. Brabec. A framework for comprehensible multi-modal detection of cyber threats, 2021.

[62] T. Komárek, J. Brabec, C. Skarda, and P. Somol. Threat hunting as a similarity search problem on multi-positive and unlabeled data. In *IEEE BigData*, pages 2098–2103. IEEE, 2021.

[63] T. Komarek, M. Grill, and T. Pevny. Passive nat detection using http access logs. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2016.

[64] T. Komárek, J. Brabec, and P. Somol. Explainable multiple instance learning with instance selection randomized trees. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II*, page 715–730, Berlin, Heidelberg, 2021. Springer-Verlag.

[65] T. Komárek, M. Grill, and T. Pevný. Detecting network address translation devices in a network based on network traffic logs, U.S. Patent 9 667 636, May 30, 2017.

[66] T. Komárek and P. Somol. End-node fingerprinting for malware detection on https data. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, New York, NY, USA, 2017. Association for Computing Machinery.

[67] T. Komárek and P. Somol. Multiple instance learning with bag-level randomized trees. In *Machine Learning and Knowledge Discovery in Databases*, pages 259–272, Cham, 2019. Springer International Publishing.

[68] T. Komárek and P. Somol. Multiple pairwise feature histograms for representing network traffic, U.S. Patent 10 867 036, Dec. 15, 2020.

[69] T. Komárek and P. Somol. Generating a vector representative of user behavior in a network, U.S. Patent 11 271 954, Mar. 8, 2022.

[70] T. Komárek and P. Somol. Instant network threat detection system, U.S. Patent 11 374 944, Jun. 28, 2022.

[71] T. Komárek, M. Vejman, and P. Somol. Training a network traffic classifier using training data enriched with contextual bag information, U.S. Patent 11 271 833, Mar. 8, 2022.

[72] M. Kopp, M. Jílek, and M. Holena. Comparing rule mining approaches for classification with reasoning. In *ITAT*, volume 2203 of *CEUR Workshop Proceedings*, pages 52–58. CEUR-WS.org, 2018.

[73] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[75] C. Leistner, A. Saffari, and H. Bischof. Miforests: Multiple-instance learning with randomized trees. In *European Conference on Computer Vision*, 2010.

[76] K. Li, R. Chen, L. Gu, C. Liu, and J. Yin. A method based on statistical characteristics for detection malware requests in network traffic. In *Third IEEE International Conference on Data Science in Cyberspace, DSC 2018, Guangzhou, China, June 18-21, 2018*, pages 527–532. IEEE, 2018.

[77] H. Lodhi and S. Muggleton. Is mutagenesis still challenging? *Proceedings Of The 15th International Conference On Inductive Logic Programming, ILP 2005, Late-Breaking Papers*, 2005.

[78] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[79] L. Machlica, K. Bartos, and M. Sofka. Learning detectors of malicious web requests for intrusion detection in network traffic, 2 2017.

[80] S. Mandlik and T. Pevny. Mapping the internet: Modelling entity interactions in complex heterogeneous networks, 2021.

[81] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[82] V. Mácha, L. Adam, and V. Šmídl. Nonlinear classifiers for ranking problems based on kernelized svm, 2020.

[83] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 490–503, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[84] K. O'Reilly and A. Brukhovetskyy. CAPE sandbox book. https://capev2.readthedocs.io/_/downloads/en/latest/pdf/, 2022. Accessed: 2023-01-14.

[85] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[86] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, Santa Clara, CA, Aug. 2019. USENIX Association.

[87] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, oct 2014. Association for Computational Linguistics.

[88] T. Pevny and M. Dedic. Nested multiple instance learning in modelling of http network traffic, 2 2020.

[89] T. Pevny and V. Kovarik. Approximation capability of neural networks on spaces of probability measures and tree-structured domains, 2019.

[90] T. Pevny and P. Somol. Discriminative models for multi-instance problems with tree structure. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, page 83–91, New York, NY, USA, 2016. Association for Computing Machinery.

[91] T. Pevný and P. Somol. Using neural network formalism to solve multiple-instance problems. In *ISNN (1)*, volume 10261 of *Lecture Notes in Computer Science*, pages 135–142. Springer, 2017.

[92] T. Pevný, V. Lisý, B. Bošanský, P. Somol, and M. Pěchouček. Explaining classifiers trained on raw hierarchical multiple-instance data, 2022.

[93] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 263–278, USA, 2016. USENIX Association.

[94] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[95] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding with unsupervised learning. 2018.

[96] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[97] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi. Microsoft malware classification challenge, 2018.

[98] K. Rose, E. Gurewitz, and G. Fox. Deterministic annealing, constrained clustering, and optimization. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pages 2515–2520 vol.3, 1991.

[99] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[100] G. Ruffo. Learning single and multiple instance decision tree for computer security applications. *University of Turin, Torino*, 2000.

[101] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[102] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, 127(1):3–30, Mar. 2011. Copyright: Copyright 2011 Elsevier B.V., All rights reserved.

[103] R. Shimizu, K. Asako, H. Ojima, S. Morinaga, M. Hamada, and T. Kuroda. Balanced mini-batch training for imbalanced image data classification with neural network. In *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, pages 27–30, 2018.

[104] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2, 2003.

[105] Stack Overflow Trends: Most popular data exchange formats. https://insights.stackoverflow.com/trends?tags=json%2Cxml%2Ccsv. Accessed: 2022-11-30.

[106] R. Starosta. Phishing detection using natural language processing. Master's thesis, Czech Technical University, 2021.

[107] J. Stiborek, T. Pevný, and M. Rehák. Multiple instance learning for malware classification. *Expert Systems with Applications*, 93:346–357, 2018.

[108] C. Straehle, M. Kandemir, U. Koethe, and F. A. Hamprecht. Multiple instance learning with response-optimized random forests. In *2014 22nd International Conference on Pattern Recognition*, pages 3768–3773, 2014.

[109] D. M. J. Tax. A matlab toolbox for multiple-instance learning, version 1.2.2, 4 2017. Faculty EWI, Delft University of Technology, The Netherlands.

[110] P. Viola, J. C. Platt, and C. Zhang. Multiple instance boosting for object detection. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[111] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[112] J. Wang and J.-D. Zucker. Solving the multiple-instance problem: A lazy learning approach. In *International Conference on Machine Learning*, 2000.

[113] W. Woof and K. Chen. A framework for end-to-end learning on semantic tree-structured data, 2020.

[114] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[115] C. Yang, M. Dong, and J. Hua. Region-based image annotation using asymmetrical support vector machine-based multiple-instance learning. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2057–2063, 2006.

[116] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[117] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal of Computer Vision*, 73(2):213–238, 2007.

[118] Q. Zhang and S. A. Goldman. Em-dd: An improved multiple-instance learning technique. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[119] Z.-H. Zhou. Multi-instance learning: A survey. *Department of Computer Science & Technology, Nanjing University, Tech. Rep*, 1, 2004.

[120] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18795–18806. Curran Associates, Inc., 2020.

[121] P. Čech, J. Kohout, J. Lokoč, T. Komárek, J. Maroušek, and T. Pevný. Feature extraction and malware detection on large https data using mapreduce. In *Similarity Search and Applications*, pages 311–324, Cham, 2016. Springer International Publishing.

[122] Šimon Mandlík, M. Račinský, V. Lisý, and T. Pevný. Jsongrinder.jl: automated differentiable neural architecture for embedding arbitrary json data. *Journal of Machine Learning Research*, 23(298):1–5, 2022.