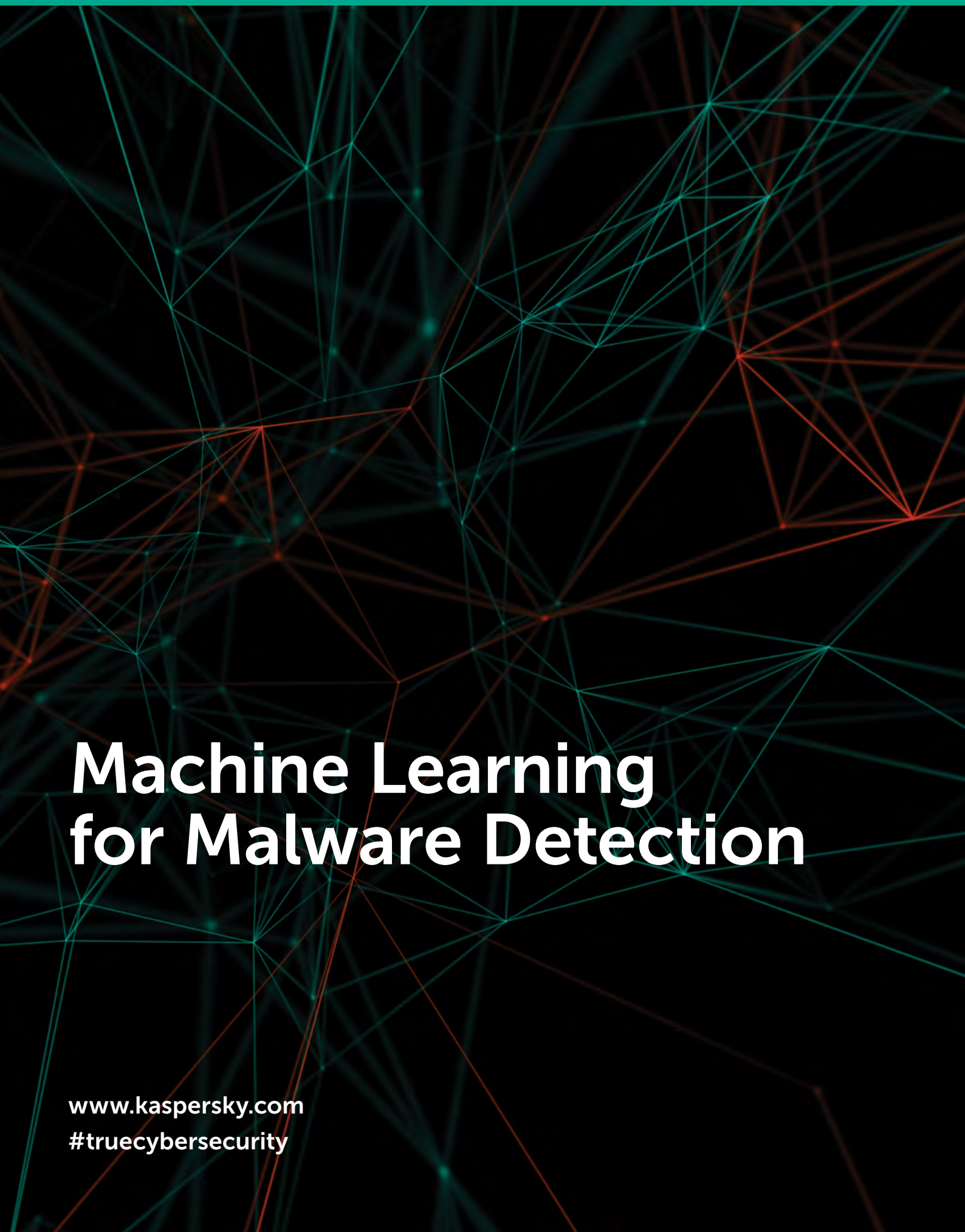
Kaspersky Enterprise Cybersecurity

KASPERSKY lab



Machine Learning for Malware Detection

www.kaspersky.com
#truecybersecurity

Contents

Machine Learning Methods for Malware Detection

In this article, we summarize our decade's worth of experience with implementing machine learning into protecting our customers from cyberthreats.

Basic approaches to malware detection

An efficient, robust and scalable malware recognition module is the key component of every cybersecurity product. Malware recognition modules decide if an object is a threat, based on the data they have collected on it. This data may be collected at different phases:

– **Pre-execution** phase data is anything you can tell about a file without executing it. This may include executable file format descriptions, code descriptions, binary data statistics, text strings and information extracted via code emulation and other similar data.

– **Post-execution** phase data conveys information about behavior or events caused by process activity in a system.

In the early epochs of the cyber era, the number of malware threats was relatively low, and simple handcrafted pre-execution rules were often enough to detect threats. But a decade ago, the tremendous growth of the malware stream did not allow anti-malware solutions to rely solely on the expensive manual creation of detection rules.

It was natural for anti-malware companies to start augmenting their malware detection and classification with **machine learning**, a computer science area that has shown great success in image recognition, searching and decision-making. Today, machine learning augments malware detection using various kinds of data on host, network and cloud-based anti-malware components.

Machine learning: concepts and definitions

According to the classic definition given by AI pioneer Arthur Samuel, machine learning is a set of methods that gives "computers the ability to learn without being explicitly programmed."

In other words, a machine learning algorithm discovers and formalizes the principles that underlie the data it sees. With this knowledge, the algorithm can reason the properties of previously unseen samples. In malware detection, a previously unseen sample could be a new file. Its hidden property could be malware or benign. A mathematically formalized set of principles underlying data properties is called the **model**.

Machine learning has a broad variety of approaches that it takes to a solution rather than a single method. These approaches have different capacities and different tasks that they suit best.

Unsupervised learning

One machine learning approach is unsupervised learning. In this setting, we are given only a data set without the right answers for the task. The goal is to discover the structure of the data or the law of data generation.

One important example is clustering. Clustering is a task that includes splitting a data set into groups of similar objects. Another task is representation learning – this includes building an informative feature set for objects based on their low-level description (for example, an autoencoder model).

Large unlabeled datasets are available to cybersecurity vendors and the cost of their manual labeling by experts is high – this makes unsupervised learning valuable for threat detection. Clustering can help with optimizing efforts for the manual labeling of new samples. With informative embedding, we can decrease the number of labeled objects needed for the usage of the next machine learning approach in our pipeline: supervised learning.

Supervised learning

Supervised learning is a setting that is used when both the data and the right answers for each object are available. The goal is to fit the model that will produce the right answers for new objects.

Supervised learning consists of two stages:

- **Training** a model and fitting a model to available training data.
- **Applying** the trained model to new samples and obtaining predictions.

The task:

- we are given a set of objects
- each object is represented with feature set X
- each object is mapped to right answer or labeled as Y

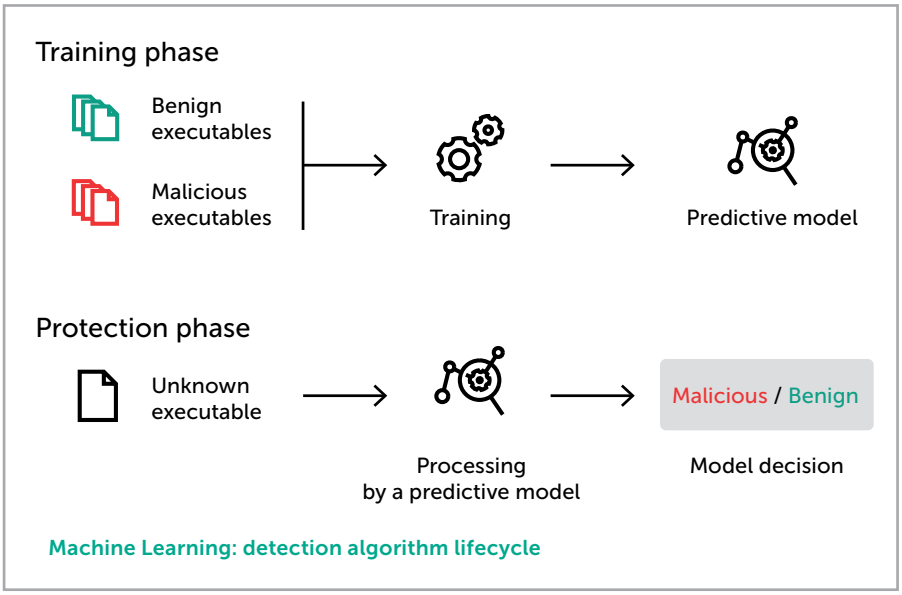
This training information is utilized during the training phase, when we search for the best model that will produce the correct label Y' for previously unseen objects given the feature set X'.

In the case of malware detection, X could be some features of file content or behavior, for instance, file statistics and a list of used API functions. Labels Y could be "malware" or "benign", or even a more fine-grained classification, such as a virus, Trojan-Downloader or adware.

In the “training” phase, we need to select some family of models, for example, neural networks or decision trees. Usually each model in a family is determined by its parameters. Training means that we search for the model from the selected family with a particular set of parameters that gives the most accurate answers for train objects according to some metric. In other words, we “learn” the optimal parameters that define valid mapping from X to Y.

After we have trained a model and verified its quality, we are ready for the next phase – applying the model to new objects. In this phase, the type of the model and its parameters do not change. The model only produces predictions.

In the case of malware detection, this is the protection phase. Vendors often deliver a trained model to users where the product makes decisions based on model predictions autonomously. Mistakes can cause devastating consequences for a user – for example, removing an OS driver. It is crucial for the vendor to select a model family properly. The vendor must use an efficient training procedure to find the model with a high detection rate and a low false positive rate.



Machine Learning: detection algorithm lifecycle

Deep learning

Deep learning is a special machine learning approach that facilitates the extraction of features of a high level of abstraction from low-level data. Deep learning has proven successful in computer vision, speech recognition, natural language processing and other tasks. It works best when you want the machine to infer high-level meaning from low-level data. For image recognition challenges, like ImageNet, deep learning based approaches already surpass humans.

It is natural that cybersecurity vendors tried to apply deep learning for recognizing malware from low-level data. A deep learning model can learn complex feature hierarchies and incorporate diverse steps of malware detection pipeline into one solid model that can be trained end-to-end, so that all of the components of the model are learned simultaneously.

Machine learning application specifics in cybersecurity

User products that implement machine learning make decisions autonomously. The quality of the machine learning model impacts the user system performance and its state. Because of this, machine learning-based malware detection has specifics.

Large representative datasets are required

It is important to emphasize the **data-driven** nature of this approach. A created model depends heavily on the data it has seen during the training phase to determine which features are statistically relevant for predicting the correct label. We will explain why making a representative data set is so important. Imagine we collect a training set, and we overlook the fact that occasionally all files larger than 10 MB are all malware and not benign, which is certainly not true for real world files. While training, the model will exploit this property of the dataset, and will learn that any files larger than 10 MB are malware. It will use this property for detection. When this model is applied to real world data, it will produce many false positives. To prevent this outcome, we needed to add benign files with larger sizes to the training set. Then, the model would not rely on an erroneous data set property.

Generalizing this, we must train our models on a data set that correctly represents the conditions where the model will be working in the real world. This makes the task of collecting a **representative dataset** crucial for machine learning to be successful.

The trained model has to be interpretable

Most of the model families used nowadays, like deep neural networks, are called **black box models**. Black box models are given the input X, and they will produce Y through a complex sequence of operations that can hardly be interpreted by a human. This could pose a problem in real-life applications. For example, when a false alarm occurs, and we would like to understand why it happened we ask: was it some problem with a training set or the model itself? The **interpretability** of a model determines how easy it will be for us to manage it, assess its quality and correct its operation.

False positive rates must be extremely low

False positives happen when an algorithm mistakes a malicious label for a benign file. Our aim is to make the false positive rate as low as possible, or “zero.” This is untypical for machine learning application. It is important, because even one false positive in a million benign files can create serious consequences for users. This is complicated by the fact that there are lots of clean files in the world, and they keep appearing.

To address this problem, it is important to impose high requirements for both machine learning models and metrics that will be optimized during training, with the clear focus on **low false positive rate (FPR) models**.

This is still not enough, because new benign files that go unseen earlier may occasionally be falsely-detected. We take this into account and implement a flexible design of a model that allows us to fix false-positives on the fly, without completely retraining the model. Examples of this are implemented in our pre- and post-execution models, which are described in following sections.

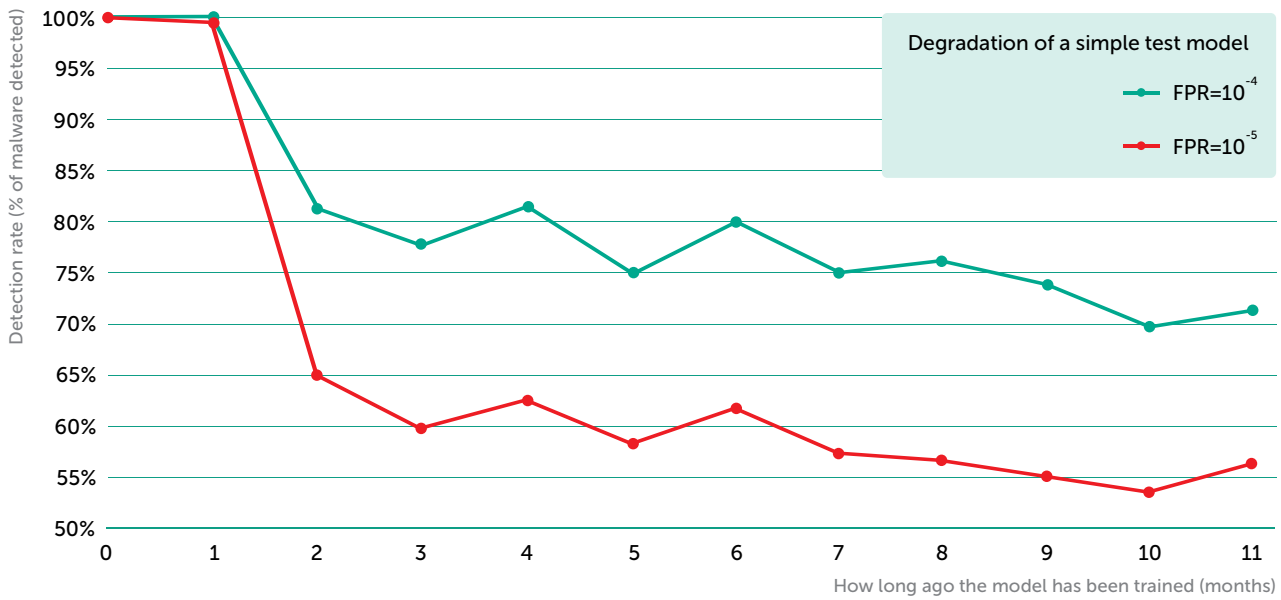
Algorithms must allow us to quickly adapt them to malware writers' counteractions

Outside the malware detection domain, machine learning algorithms regularly work under the assumption of **fixed data distribution**, which means that it doesn't change with time. When we have a training set that is large enough, we can train the model so that it will effectively reason any new sample in a test set. As time goes on, the model will continue working as expected.

After applying machine learning to malware detection, we have to face the fact that our data distribution isn't fixed:

- Active adversaries (malware writers) constantly work on avoiding detections and releasing new versions of malware files that differ significantly from those that have been seen during the training phase.
- Thousands of software companies produce new types of benign executables that are significantly different from previously known types. The data on these types was lacking in the training set, but the model, nevertheless, needs to recognize them as benign.

This causes serious changes in data distribution and raises the problem of detection rate degradation over time in any machine learning implementation. Cybersecurity vendors that implement machine learning in their antimalware solutions face this problem and need to overcome it. The architecture needs to be flexible and has to allow model updates "on the fly" between retrainings. Vendors must also have effective processes for collecting and labeling new samples, enriching training datasets and regularly retraining models.



Machine Learning: test model detection rate degradation over time

Kaspersky Lab machine learning application

The aforementioned properties of real world malware detection make straightforward application of machine learning techniques a challenging task. Kaspersky Lab has almost a decade's worth of experience when it comes to utilizing machine learning methods in information security applications.

Detecting new malware in pre-execution with similarity hashing

At the dawn of the antivirus industry, malware detection on computers was based on heuristic features that identified particular malware files by:

- code fragments
- hashes of code fragments or the whole file
- file properties
- and combinations of those features.

The main goal was to create a reliable fingerprint—a combination of features – of a malicious file that could be checked quickly. Earlier, this workflow required the manual creation of detection rules, via the careful selection of a representative sequence of bytes or other features signaling a malware. During the detection, an antiviral engine in a product checked the presence of the malware fingerprint in a file against known malware fingerprints stored in the antivirus database.

However, malware writers invented techniques like server-side polymorphism. This resulted in the flow of hundreds of thousands malicious samples discovered on a daily basis. At the same time, the fingerprints used were sensitive to small changes in files. Minor changes in existing malware took it off the radar. The previous approach quickly became ineffective because:

- Creating detection rules manually didn't keep up with the emerging flow of malware.
- Checking each file's fingerprint against a library of known malware meant that you couldn't detect new malware until analysts manually create a detection rule.

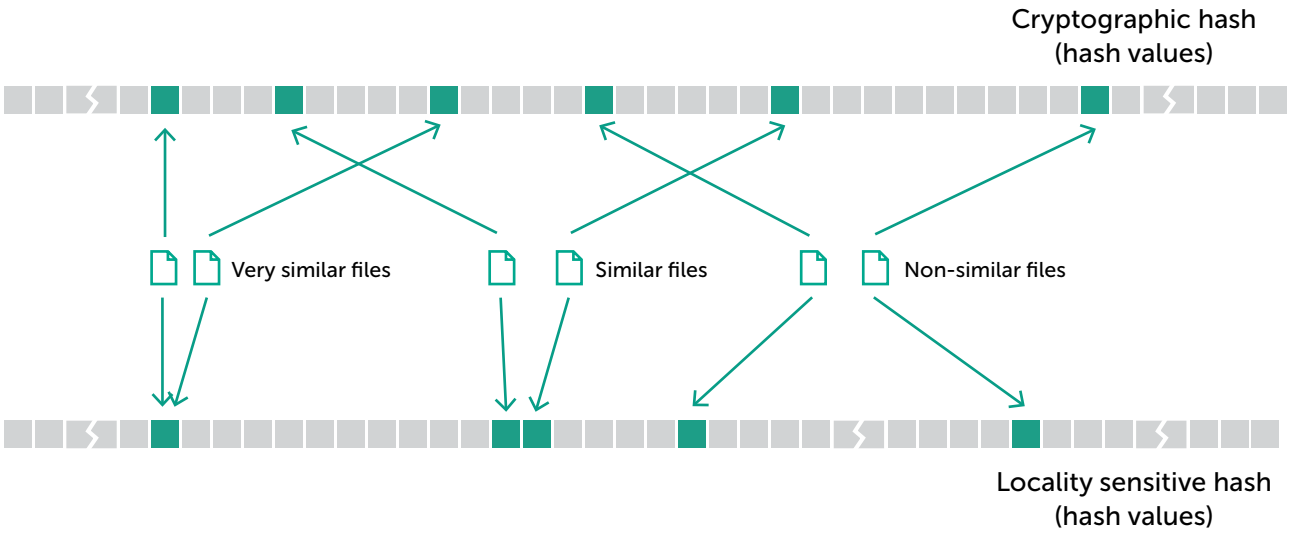
We were interested in features that were robust against small changes in a file. These features would detect new modifications of malware, but would not require more resources for calculation. Performance and scalability are the key priorities of the first stages of anti-malware engine processing.

To address this, we focused on extracting features that could be:

- calculated quickly, like statistics derived from file byte content or code disassembly,
- directly retrieved from the structure of the executable, like a file format description.

Using this data, we calculated a specific type of hash functions called locality-sensitive hashes (LSH).

Regular cryptographic hashes of two almost identical files differ as much as hashes of two very different files. There is no connection between the similarity of files and their hashes. However, LSHs of almost identical files map to the same binary bucket – their LSHs are very similar – with high probability. LSHs of two different files differ substantially.



Machine Learning: locality sensitive hashing

But we went further. The LSH calculation was unsupervised. It didn't take into account our additional knowledge of each sample being malware or benign.

Having a dataset of similar and non-similar objects, we enhanced this approach by introducing a training phase. We implemented a **similarity hashing** approach. It's similar to LSH, but it is supervised and capable of utilizing information about pairs of similar and non-similar objects. In this case:

- Our training data X would be pairs of file feature representations [X1, X2]
- Y would be the label that would tell us whether the objects were actually semantically similar or not.
- During training, the algorithm fits parameters of hash mapping h(X) to maximize the number of pairs from the training set, for which h(X1) and h(X2) are identical for similar objects and different otherwise.

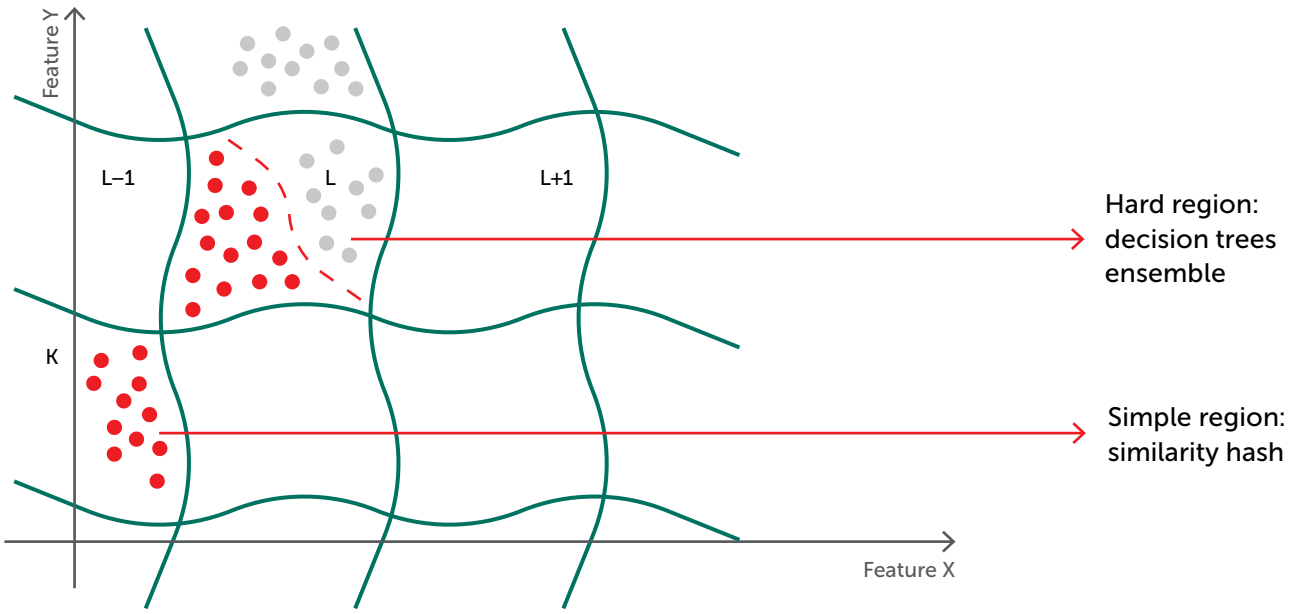
This algorithm that is being applied to executable file features provides specific similarity hash mapping with useful detection capabilities. In fact, we train several versions of this mapping that differ in their sensitivity to local perturbations of different sets of features. For example, one version of similarity hash mapping could be more focused on capturing the executable file structure, while paying less attention to the actual content. The other one could be more focused on capturing the ASCII-strings of the file.

This captures the idea that different subsets of features could be more or less discriminative to different kinds of malware files. For one of them, file content statistics could reveal the presence of an unknown malicious packer. For the others, the most important piece of information regarding potential behavior is concentrated in strings representing used OS API, created file names, accessed URLs or other feature subsets.

For more precise detection in products, the results of a similarity hashing algorithm are combined with other machine learning-based detection methods.

Two-stage pre-execution detection on users' computers with similarity hash mapping combined with decision trees ensemble

To analyze files during the pre-execution stage, our products combine a similarity hashing approach with other trained algorithms in a two-stage scheme. To train this model, we use a large collection of files that we know to be malware and benign.



Machine Learning: segmentation of object space

Schematic representation of segmentation of the object-space created with similarity hash mapping. For simplicity, the illustration has only two dimensions. An index of each cell corresponds to the particular similarity hash mapping value. Each cell of the grid illustrates a region of objects with the same value of similarity hash mapping, also known as a hash bucket. Dot colors: malicious (red) and benign (blue). Two options are available: add the hash of a region to the malware database (simple regions) or use it as the first part of the two-stage detector combined with a region-specific classifier (hard regions).

The two-stage analysis design addresses the problem of reducing computational load on a user system and preventing false positives.

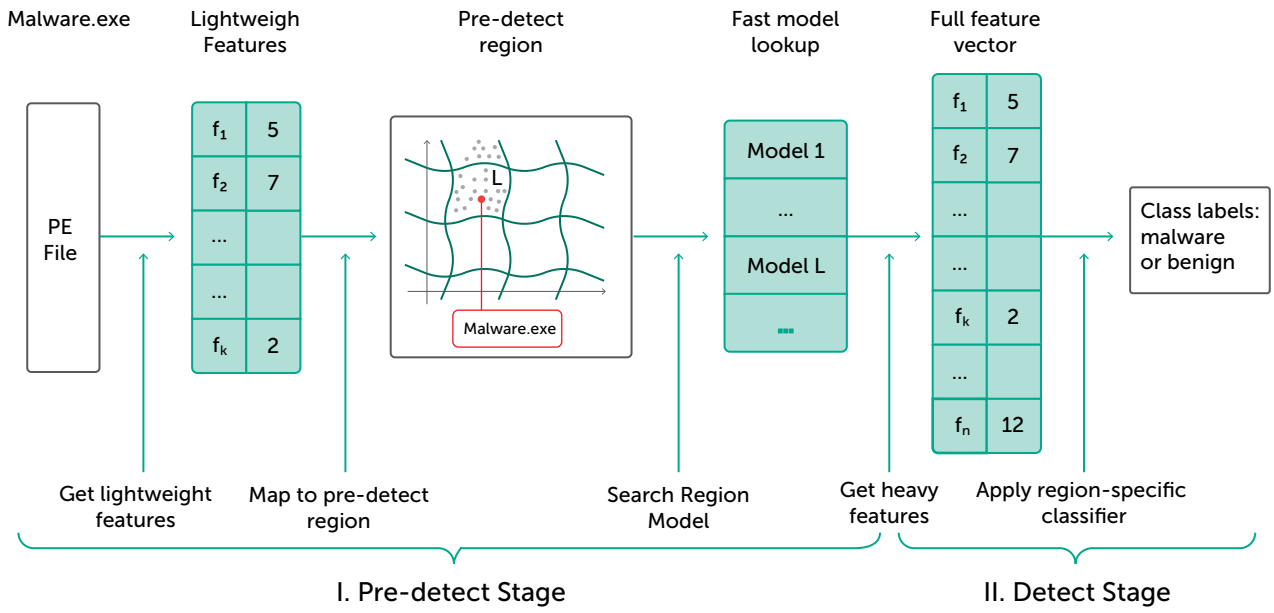
Some file features important for detection require larger computational resources for their calculation. Those features are called "heavy". To avoid their calculation for all scanned files, we introduced a preliminary stage called a **pre-detect**. A pre-detect occurs when a file is analyzed with "lightweight" features and is extracted without substantial load on the system. In many cases, a pre-detect provides us with enough information to know if a file is benign and ends the file scan. Sometimes it even detects a file as malware. If the first stage was not sufficient, the file goes to the second stage of analysis, when "heavy" features are extracted for precise detection.

In our products, the two-stage analysis works in the following way. In the pre-detect stage, learned similarity hash mapping is calculated for the lightweight features of the scanned file. Then, it's checked to see if there are any other files with the same hash mapping, and whether they are malware or benign. A group of files with a similar hash mapping value is called a **hash bucket**. Depending on the hash bucket that the scanned file falls into, the following outcomes may happen:

- In a **simple region** case, the file falls into a bucket that contains only one kind of object: malware or benign. If a file falls into a "pure malware bucket" we detect it as malware. If it falls to a "pure benign bucket" we don't scan it any deeper. In both cases, we do not extract any new "heavy" features.

- In a **hard region**, the hash bucket contains both malware and benign files. It is the only case when the system may extract "heavy" features from the scanned file for precise detection. For each hard region, there is a separate region specific classifier trained. Currently we use a modification of decision trees ensemble or a "heavy" feature-based similarity hashing, depending on what is more effective in each hard region.

In reality, there are some hard regions that are not suitable for further analysis by this two-stage technology, because they contain too many popular benign files. Processing them with this method yields a high risk of false positives and performance degradation. For such cases, we do not train a region specific classifier and do not scan files in this region through this model. For correct analysis in a region like this we use other detection technologies.



Machine Learning: two-stage classifier

Implementation of a pre-detect stage drastically reduces the amount of files that are heavily-scanned during the second step. This process improves the performance because the lookup by similarity hash mapping in the pre-detect phase is completed quickly.

Our two-stage design also reduces the risk of false positives:

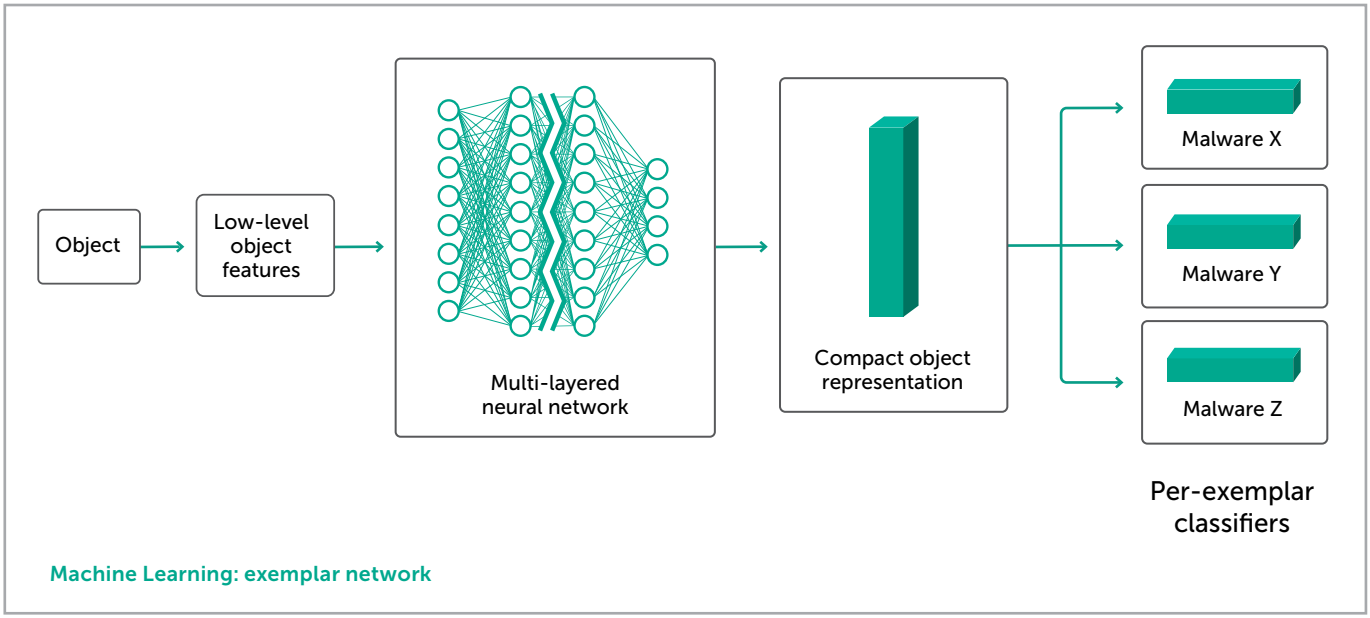
- In the first (pre-detect) stage, we do not enable detection with region specific classifiers in regions with a high risk of false positives. Because of this, the distribution of objects passed to the second stage is biased towards the "malware" class. This reduces the false positive rate, too.
- In the second stage, classifiers in each hard region are trained on malware from only one bucket—but on all clean objects available in all the buckets of the training set. This makes a regional classifier detect the malware of a particular hard region bucket more precisely. It also prevents any unexpected false positives, when the model works in products with real-world data.

Interpretability of the two-stage model comes from the fact that each hash in a database is associated with some subset of malware samples in training. The whole model could be adapted to a new daily malware stream via adding detections, including hash mappings and tree ensemble models for a previously unobserved region. This lets us revoke and retrain region specific classifiers without significantly degrading the detection rate of the whole product. Without this, we would need to retrain the whole model on all of the malware that we know with every change we would want to make.

That being said, the two-stage malware detection is suitable for the specifics of machine learning that were discussed in the introduction.

Deep learning against rare attacks

Typically, machine learning faces tasks when malicious and benign samples are numerously represented in the training set. But some attacks are so rare that we have only one example of malware for training. This is typical for high-profile targeted attacks. In this case, we use a very specific deep learning-based model architecture. We call this approach **exemplar network (ExNet)**.



Machine Learning: exemplar network

The idea here is that we train the model to build **compact representations** of input features. We then use them for the to simultaneously train multiple **per-exemplar classifiers** – these are algorithms that detect particular types of malware. Deep learning allows us to combine these multiple steps (object feature extraction, compact feature representation and local, or per-exemplar, model creation) into one neural network pipeline that distills the discriminative features for various types of malware.

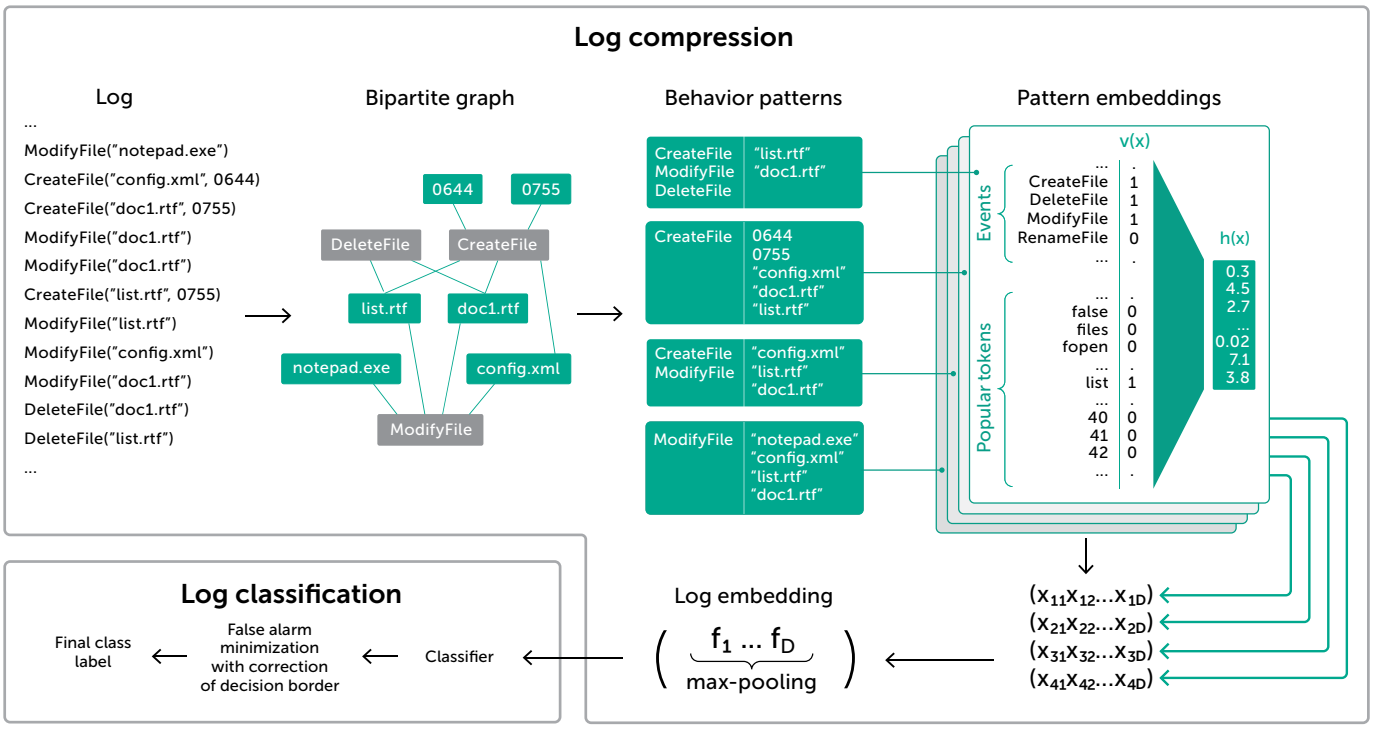
This model can efficiently generalize knowledge about single malware samples and a large collection of clean samples. Then, it can detect new modifications of corresponding malware.

Deep learning in post-execution behavior detection

The approaches described earlier were considered in the framework of static analysis, when an object description is extracted and analyzed before the object's execution in the real user environment.

Static analysis at the pre-execution stage has a number of significant advantages. The main advantage is that it is safe for the user. An object can be detected before it starts to act on a real user's machine. But it faces issues with advanced encryption, obfuscation techniques and the use of a wide variety of high-level script languages, containers, and fileless attack scenarios. These are situations when post-execution behavior detection comes into play.

We also use deep learning methods to address the task of behavior detection. In the post-execution stage, we are working with behavior logs provided by the threat behavior engine. The behavior log is the sequence of system events occurring during the process execution, together with corresponding arguments. In order to detect malicious activity in observed log data, our model compresses the obtained sequence of events to a set of binary vectors. It then trains a deep neural network to distinguish clean and malicious logs.



Machine Learning: behavior model pipeline

A log's compressing stage includes several steps:

1. The log is transformed into a **bipartite behavior graph**. This graph contains two types of vertices: events and arguments. Edges are drawn between each event and argument, which occur together in the same line in the log. Such a graph representation is much more compact than the initial raw data. It stays robust against any permutations of lines caused by tracing different runs of the same multiprocessing program, or behavior obfuscation by the analyzed process.
2. After that, we automatically extract specific subgraphs, or **behavior patterns**, from this graph. Each pattern contains a subset of events and adjacent arguments related to a specific activity of the process, such as network communications, file system exploration, modification of the system register, etc.
3. We compress each "behavior pattern" to a sparse binary vector. Each component of this vector is responsible for the inclusion of a specific event or argument's token (related to web-, file- and other types of activity) in the template.
4. The trained deep neural network transforms sparse binary vectors of behavior patterns into compact representations called **pattern embeddings**. Then they are combined into a single vector, or **log embedding**, by taking the element-wise maximum.
5. Finally, based on the log embedding, the network predicts the log's suspiciousness.

The main feature of the used neural network is the positiveness of all the weights and the monotony property of all the activation functions. These properties provide us with many important advantages:

- Our model's suspicion score output only grows with time while processing new lines from the log. As a result, malware cannot evade detection by performing additional noise or a "clean" activity in parallel with its main payload.
- Since the model's output is stable in time, we are probably protected from eventual false alarms caused by the prediction's fluctuation in the middle of scanning of a clean log.
- Working with log samples in a monotonic space allows us to automatically select events that cause the detection and manage false alarms more conveniently.

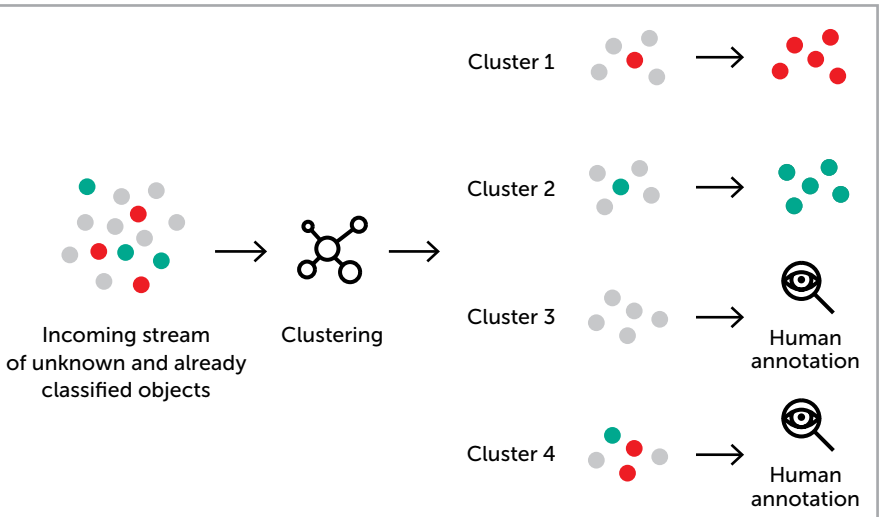
An approach like this enables us to train a deep learning model capable of operating with high-level interpretable behavior concepts. This approach is safe being applied to the whole diversity of user environments and incorporates false alarm fixing capabilities in its architecture. Together, all of that gives us a powerful mean for the behavioral detection of the most complicated modern threats.

Applications in the infrastructure

From efficiently processing incoming streams of malware in Kaspersky Lab to maintaining large-scale detection algorithms, machine learning plays an equally important role when it comes to building a proper in-lab infrastructure.

Clustering the incoming stream of objects

With hundreds of thousands samples coming in to Kaspersky Lab every day along with the high cost of manual annotation of new types of samples, reducing the amount of data that analysts would need to look at becomes a crucial task. With efficient clustering algorithms, we can go from an unbearable number of separate unknown files to a reasonable number of object groups. Parts of these object groups would be automatically processed based on the presence of an already annotated object inside it.



Machine learning: clustering the incoming stream of objects

All recently received incoming files are analyzed by our in-lab malware detection techniques including pre- and post-execution. We aim to label as many objects as possible, but some objects are still unclassified. We want to label them. For this, all objects, including the labeled ones, are processed by multiple feature extractors. Then, they are passed together through several clustering algorithms (e.g. K-means and dbscan) depending on the file type. This produces groups of similar objects.

At this point, we face four different types of resulting clusters with unknown files:

1) clusters that contain malware and unknown files;
2) clusters that contain clean and unknown files;
3) clusters that contain malware, clean and unknown files;
4) cluster that only contain unknown files.

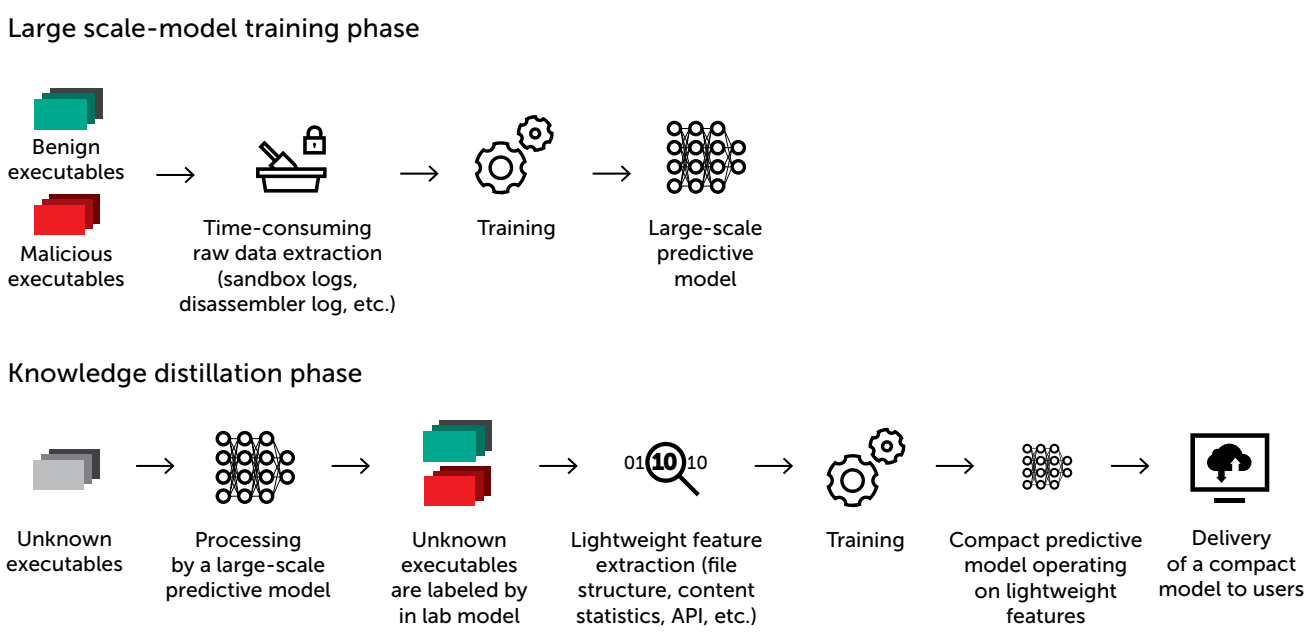
For objects in the clusters of types 1-3, we use additional machine learning algorithms like belief propagation to verify the similarity of unknown samples to classified ones. In some cases this is effective even in the clusters of type 3. This allows us to automatically label unknown files, leaving only the clusters of type 4, and partially of type 3, for humans. This results in the drastic reduction of human annotations needed on a daily basis.

Distillation: packing the updates

The way we detect malware in-lab is different from algorithms optimal for user products. Some of the most powerful classification models require a large amount of resources like CPU/GPU time and memory, along with expensive feature extractors.

For example, since most of the modern malware writers use advanced packers and obfuscators to hide payload functionality, machine learning models will highly benefit from using execution logs from an in-lab sandbox with advanced behavior logging. At the same time, gathering these kinds of logs in a pre-execution phase on a user's machine could be computationally intense. It could result in notable user system performance degradation.

It is more effective to keep and run those "heavy" models in-lab. Once we know that a particular file is malware, we use the knowledge we have gained from the models to train some lightweight classifier that is going to work in our products.



Machine Learning: large scale model distillation

In machine learning, this process is called **distillation**. We use it to teach our products to detect new kinds of malware:

1. In our lab, we first extract some time-consuming features from labeled files and train a "heavy" in-lab model on them.
2. We take a pool of unknown files and use our "heavy" in-lab model to label them.
3. Then, we use the newly labeled files to augment the training set for the lightweight classification model.
4. We deliver the lightweight model to user products.

Distillation allows us to effectively export our knowledge on new and unknown threats to our users.

What did we learn about machine learning after doing it for a decade?

We learned that passing the routine to an algorithm leaves more space for us to research and create. This allows us to deliver better protection to our customers. Through our attempts, failures and wins, we eventually learned what is important when it comes to letting machine learning make its superior impact into malware detection.

Here is the boilerplate:

- **Have the right data.** This is the fuel of machine learning. The data must be representative, relevant to current malware landscape and correctly labeled when needed. We became experts in extracting and preparing data and training our algorithms. We made a sufficient collection with some billions of file samples to empower machine learning.

- **Know theoretical machine learning and how to apply it to cybersecurity.** We understand how machine learning works in general and keep track of state-of-the-art approaches emerging in the field. On the other hand, we are also experts in cybersecurity and we foresee the value each innovative theoretical approach brings to cybersecurity practices.

- **Know user practical needs and be an expert at implementing machine learning** into products that help users with their needs. We make machine learning work effectively and safely. We build innovative solutions that are largely required by the cybersecurity market.

- **Earn a sufficient user base.** This introduces the power of "crowdsourcing" to detection quality and gives us the feedback we need to let us know if we are right or wrong.

- **Keep detection methods in multi-layered synergy.** As long as today's advanced threat attack vectors are very diverse, cybersecurity solutions should deliver protection at multiple layers. In our products, machine learning-based detection works synergistically with other kinds of detection in a multi-layered arsenal of modern cybersecurity protection.

Kaspersky Lab
Enterprise Cybersecurity: **www.kaspersky.com/enterprise**
Cyber Threats News: **www.securelist.com**
IT Security News: **business.kaspersky.com/**

#truecybersecurity
#HuMachine

www.kaspersky.com