January 2020

ROBUST DECISION FOREST INFERENCE ALGORITHM IN PRESENCE OF MISSING FEATURE VALUES

Jan Brabec

Cenek Skarda

ROBUST DECISION FOREST INFERENCE ALGORITHM IN PRESENCE OF MISSING FEATURE VALUES

AUTHORS:

Jan Brabec
Cenek Skarda

ABSTRACT

Presented herein is a novel algorithm for inference on decision forest models that increases the robustness of the decisions in the presence of missing features in the data. The proposed algorithm ensures that tree decisions are supported by a minimal amount of non-missing features. Experiments have demonstrated that the proposed algorithm not only increases the robustness of the model, but also increase the model's predictive performance.

DETAILED DESCRIPTION

Artificial Intelligence (AI) systems often combine many different sources of data that are assembled to a single feature vector for an machine learning (ML) classifier, such as random forest, to make a prediction. This situation exists in many different domains. For example, many different sources of information may be used to determine if a single packet flow is related to malware. The sources of information include, for example:

- Features extracted directly from the packet flow directly (e.g., duration, size).
- If available, ETA features.
- If available for a given IP, features from passiveDNS.
- If available, information from the TLS certificate.
- etc.

Many of the features may be unavailable for a given packet flow, but there is still a desire to build a general classifier that is able to utilize all of the information (when available), while remaining robust when some of the features are missing. For such situations, classifiers based on decision trees are good candidates as they are able to handle missing values quite naturally. The problem is that without any guard rails, in some situations, the decision of the classifier may be supported by only a handful of non-missing features. This makes the classifier non-robust and very vulnerable to noisy or corrupted features. As the data pipelines in modern ML systems are very complex, it is only a matter of time when a bug or data corruption (it may be in a data source under 3rd party control) breaks the classifier which can have serious impacts.

Accordingly, presented herein is a new algorithm for inference over decision forests that greatly mitigates the risk that a small number of corrupted or noisy features will impact the prediction of the forest.

Before describing aspects of the proposed algorithm, a brief description of the structure of decision forests and how the standard inference algorithm works is first provided. In particular, a decision forest is an ensemble of decision trees and Figure 1, below, shows a common instance of a single decision tree with five (5) nodes.

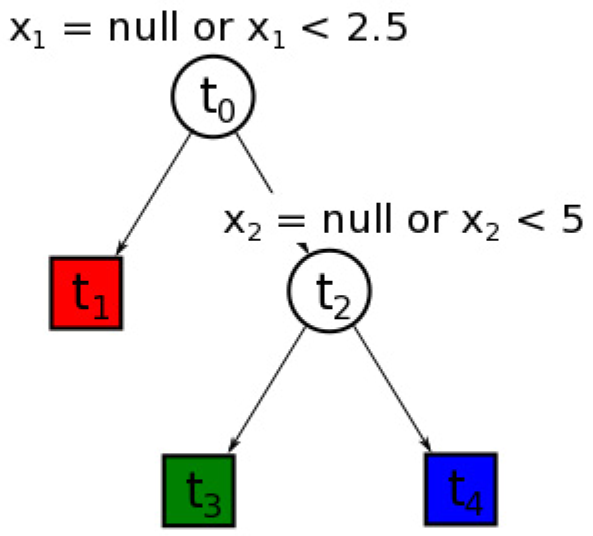

Figure 1

A decision tree such as that shown in Figure 1 can be inducted with any known algorithm for training decision trees. The null checks that are explicit in Figure 1 are usually accomplished by substituting missing values (null) with very low values, such as -infinity. In this way, the algorithm itself does not need to handle missing values in any specific fashion. When classifying a feature vector "x," the standard prediction algorithm starts at the root node (t_0) and evaluates the associated rule (x_1 = null or $x_1 < 2.5$). If the result is true, then it descends to the left subtree (and predicts red class). However, if it is false, it recurses into the right subtree.

The above procedure repeats independently for every tree in the forest and the final prediction of the forest is accomplished by aggregating the votes from individual trees. This is usually done by simple majority voting.

In the context of the techniques presented herein, a model is deemed to be non-robust if the performance of the model can be strongly affected by changes to a very small number of features. In a production system, this can happen for various reasons and most of the reasons cannot be foreseen. For example, the features may change due to changes in the computation pipeline of that particular feature, or due to changes to the underlying data from which the feature is computed. Often, these changes are not under the control of the researcher that designed the end classifier.

During the descent to the correct leaf for input vector x, the standard inference algorithm can encounter any number of missing values. The path can be arbitrarily long, but it is not uncommon for most of the conditions to be satisfied because the value for their specific feature is missing. When investigating incidents, it was determined that often there are vectors classified only based on a single present feature value, even if the length of the path from the root to the leaf was more than 20. This makes the classifier very sensitive to that particular feature and error-prone because of unforeseeable ways how the single feature value could change in time.

It is to be noted that, with a binary decision tree of the type as in Figure 1, it is not possible to discover offline whether a particular node in the tree structure is more sensitive to missing values. For example, in the root node t_0, both $x_1 = 1$ and x_1 = null would cause descent to the left subtree, thus it is not possible to prune out the paths that would result in non-robust decisions from the model.

It is also noted that a value that is missing for a feature may be informative and it may be possible to use this fact in the decision making (e.g., the feature related to the content of a website may be missing, because the site is no longer available). However, if most of the features are missing yields, then a decision is not based on any knowledge at all.

The techniques presented herein extend the inference part of the algorithm (not the training) to make sure that, if the classifier makes a decision, then it is based on support from multiple non-missing features. This reduces the sensitivity of the classifier to unexpected changes in individual features. If the decision for the current sample that is being classified is not sufficiently robust, then a default label is returned. In the case of classification of network traffic for malware, the default label may be 'LEGIT' because the imbalance between classes makes the LEGIT class the most probable and because positive detections have to be explainable, which is difficult if most of the feature values are missing.

The proposed inference algorithm has three (3) parameters:

1. min_non_missing_features
2. min_votes
3. default_label

The proposed algorithm has a similar structure to the standard inference algorithm, but has been modified to be aware of missing values. The inference in each decision tree is modified so that, during descending to the leaf, the algorithm keeps count (in variable seen_non_missing) how many times it has based its decision on a feature value that is not missing. In the leaf, if seen_non_missing is greater or equal to min_non_missing_features, then the standard leaf prediction is returned. Otherwise, nothing is returned. Detailed pseudocode for the tree inference is shown below in Figure 2 (Algorithm 1).

Algorithm 1 Decision tree inference robust to missing values. The function returns either the prediction of the tree or nothing if not enough non-missing values were encountered on the path to the leaf.

```
function TreePredict($\mathbf{x}$, node, t, seen_non_missing = 0)
    if node.is_leaf() then
        if seen_non_missing $\geq$ t then
            return node.prediction
        else
            return null
    else
        if $\mathbf{x}$(node.split_dimension) $\neq$ null then
            seen_non_missing++
        if node.should_go_left($\mathbf{x}$) then
            return TreePredict($\mathbf{x}$, node.left_child, t, seen_non_missing)
        else
            return TreePredict($\mathbf{x}$, node.right_child, t, seen_non_missing)
```

Figure 2

The inference on the level of decision forest is modified, as described below. In particular, the forest first collects all votes from its trees (each tree can either vote for a single class or not at all). If the number of all votes is higher than the min_votes parameter, then the class that received the majority of the votes is returned as a prediction of the forest. If there are not enough votes, then the default label is returned. Detailed pseudocode for the forest inference is available is shown below in Figure 3 (Algorithm 2).

Algorithm 2 Decision forest majority voting inference robust to missing values. If not enough trees vote then default label is returned.

```
function ForestPredict(x, forest, min_non_missing_features, min_votes, default_label)
    votes = []
    for each tree $\in$ forest do
        vote = TreePredict(x, tree, min_non_missing_features, 0)
        if vote $\neq$ null then
            votes.add(vote)
    if votes.size $\geq$ min_votes then
        return MajorityVote(votes)
    else
        return default_label
```

Figure 3

As noted, the techniques presented herein only describe the prediction phase of the decision forest algorithm. Any known algorithm for training of decision forests can still be used for training.

Experiments have been conducted to train a decision forest model on a subset of proxy logs related to both encrypted and unencrypted traffic originating from telemetry data between 3.3.2019 and 20.4.2019. The training dataset contained almost 139 million proxy logs. The data was used to evaluate the proposed algorithm against a standard inference scheme on seven (7) days of proxy logs between 22.4.2019 and 28.4.2019. The test dataset contained around 1.4 billion proxy logs where 159 classes related to malware were identified. The number of trees in the Random Forest (RF) was 70.

In these experiments, the parameters for missing values aware RF inference were:

1. min_non_missing_features = 5
2. min_votes = 35
3. default_label = LEGIT

The following table shows the aggregated results of the experiment:

	Precision	Recall
Standard RF inference	98.5 %	64.9 %
Missing Values Aware RF inference (Proposed Algorithm)	98.5 %	68.5 %

Prior to the above experiment, the expectation was that with the proposed algorithm result in a slight decrease in performance which would be a tradeoff for the increased robustness of the results and stability over a longer time frame in a production environment. However, as shown in the above table, the proposed algorithm actually has very similar precision and superior recall compared to the baseline. This optimistic result suggests that ignoring decisions of trees that were based on a very small number of feature values can help not only as a guard rail against breaking the classifier via a misbehaving feature, but also helps improve general predictive performance.