IOT Device Identification Using Machine Learning Techniques

Idan Mosseri, Dean Eckert

# **Introduction**

The “Internet of Things” (IOT) relates to networks of physical devices and items embedded with electronics, sensors, actuators, software and connectivity which enables the communication between these devices and their exchange of data.

In recent years, more organizations allow IOT devices to be connected to their networks which might impose a security threat to these networks. Hence, organizations must be able to identify which devices are connected to their networks and whether these devices are considered legitimate and do not impose a risk.

Leveraging network traffic in order to identify devices in general has been gaining in popularity in previous works. Specifically, there is an increasing interest in the domain of IOT device identification due to the importance of identifying such devices in an organizational environment (especially in terms of security).

In this work, we address the challenge of identifying an IOT device by analyzing its high-level network traffic data using machine learning techniques. We would like to develop a method for identifying such device, even if its IP address has been spoofed (which can be done easily) and to allow us identify an abnormal behaviour which may indicate which device is in-use.

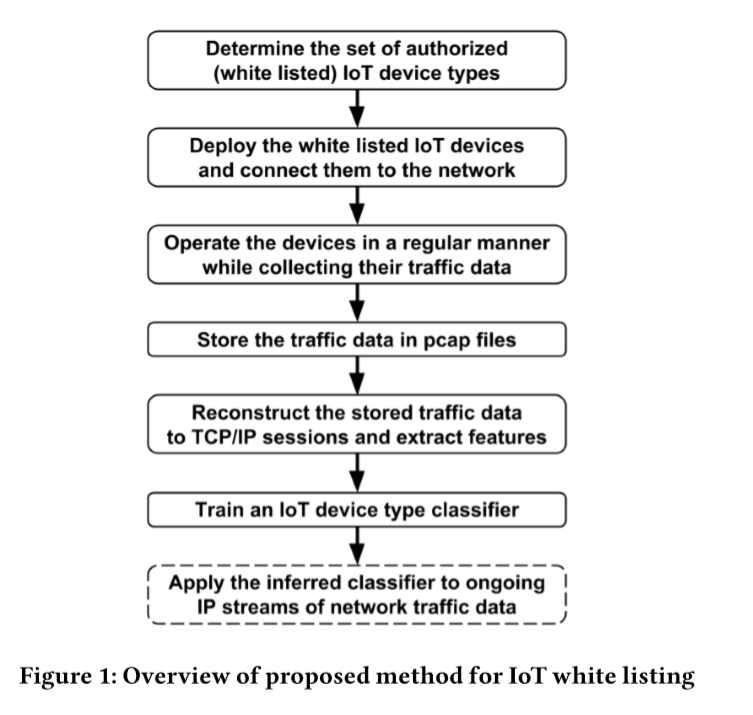
Since we can’t rely on the IP address as an identifier of the device (since this value can be spoofed), we would like to analyze the traffic’s high-level data (which means the metadata and traffic statistics, rather than analyzing the content).

The challenge we would like to address in this research is essentially a multi-class problem. We will use a dataset collected from 10 different IOT devices. The dataset contains information about the network traffic of these devices. The approach we will use in this research will try to predict which device is it according to a given traffic session or sequence of sessions. We will start by creating one-vs-rest classifiers for each device and we will go on until we will be able to distinguish between all the different devices.

# **Related Works**

Previous work has already been conducted about the topic of identifying an IOT device in an organizational environment. The most relevant work in this area is described in a paper about a system called ProfilIOT [1]. The paper describes an experimental environment in which network traffic data was collected from different nine devices (where seven of them were IOT devices). Following the data collection stage, the data was transformed into the form of sessions (TCP connection from its SYN packet to the FIN packet) and a meta-classifier was trained and evaluated. The meta-classifier used session-based classifier which predicts the probability that a given session was originated from this specific device. Following a definition of a threshold, the classifier was also being able to predict whether it was actually originated from the device or not. Eventually, the meta-classifier used these device-specific classifiers in order to predict the device in which a sequence of sessions was originated from. This work used some specific machine learning models like Random Forest, GBM and XGBoost.

In another paper [2], a similar mechanism was used for the detection of unauthorized IOT devices in the network using machine learning techniques. Traffic was collected from 17 distinct IOT devices, representing 9 types of IOT devices. Based on a classifications of a 20-sessions sequences and the use of the majority rule the classifier managed to understand whether the device was part of the whitelisted devices. The machine learning model used in this paper was the Random Forest model. The general method proposed in the paper as follows:



As can be seen in the diagram above, the whitelisted devices should be defined in advance, learn the “normal” behavior of these devices and training a relevant classifier. Following this training stage, the classifier will be able to determine whether the next ongoing IP streams are of a legitimate device or not.

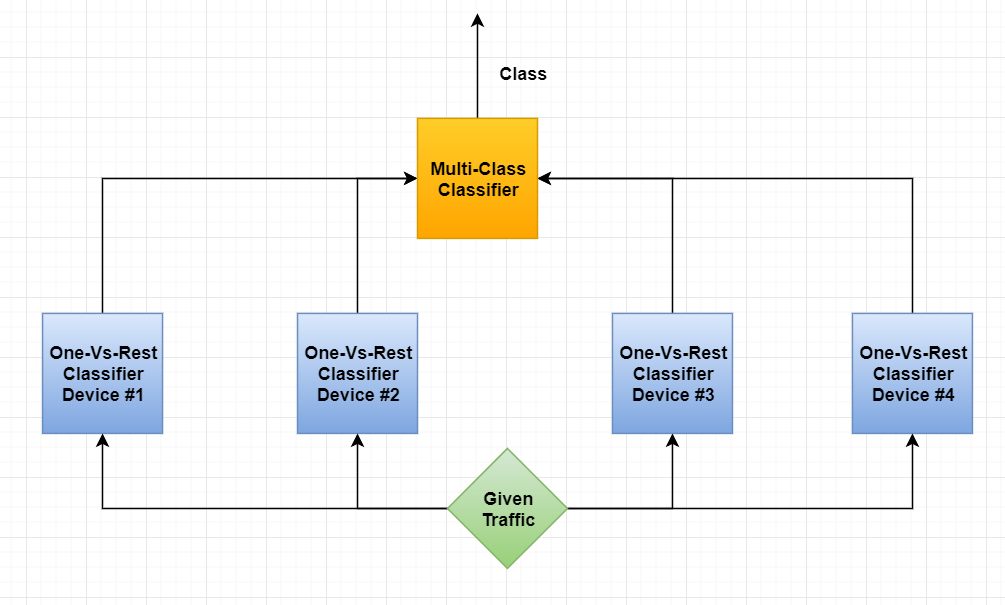
It is also worth mentioning that there are previous works related to the usage of network traffic data analysis in general and not only in relation to the IOT domain. An example for such work is a paper describing a method of unknown malware detection using network traffic classification [3]. In this paper, a method using a supervised machine learning techniques is described and the way it can be used for detection of malicious communication like the interaction with a command and control servers. The solution was based on different layers of the network stack and different protocols. The paper is useful for understanding the different approaches of network data analysis like packet-level vs flow-level analysis, port-based attributes vs payload based attributes vs statistical based attributes etc.

# **Proposed Method**

The chosen method for this research is based on the method described in the paper about ProfilIOT. As presented in the paper, we would like to perform the research in stages:

The first stage will be focused on creating One-Vs-Rest classifiers for each device using a given training set. We will examine different questions related to different models and their parameters and we would like to answer the question of whether it is possible to create such classifiers with high accuracy, but also resistant to possible effects of imbalanced data. First, the classifiers will be created at the level of a single session (which means that each classifier will get as its input one instance from the dataset). However, we would also like to examine the question of whether it is possible to use multiple-session sequences to achieve higher accuracy and AUC scores.

The second stage will be focused on the task of multi-class classification. We will try to create a classifier which will be able to get one vector representing a session (or several sessions as a sequence) and determine which device from the 10 given devices originated this traffic.



For each stage we will try to answer its relevant research questions in order to optimize the models and understanding which parameters and criteria have the most impact on the performance of the classifier. We will try to understand whether it is possible to use “rule-of-thumb” for choosing the model or whether it must be specific to a given device. We will compare the analysis at the level of a single session with multiple-sessions sequences and we will address both problems of one-vs-rest classifiers and multi-class classifiers.

The general algorithm will be as follows:

1. For each device d(i):
   1. Create a classifier c(i) which had best results according to the experiments. The classifier will be of the form of a one-vs-rest classifier
   2. Add the classifier to classifiers\_list
2. Given a new session or sequence of sessions:
   1. For each classifier c(i) in classifiers\_list (sequential scanning of the list):
      1. Get classification - > class(i)
      2. If class(i) == True (which means this is the device):
         1. Return class(i)
      3. Else:
         1. Continue to the next c(i)
   2. If we reached here and no classification was True:
      1. Return random device

Note that we will also try to find the optimal sequence length as described in the ProfilIOT paper using a similar algorithm.

**First Stage: One-Vs-Rest Classifiers**

In this stage, we will create several classifiers by using different models and different parameters. We would like to find the best parameters and models in order to create classifiers for each device which will determine whether the given traffic (in the form of a single session) was originated from this device or not. We will examine whether average values can indicate about the best models or whether the trained model should be device-specific. In this section, we will try to answer the following research questions:

1. Which models tend to have more accuracy and better AUC values for this task on average?
2. What makes Decision Tree classifiers more accurate and with better AUC values? Is it the splitting criterion (gini or entropy)? Is it the minimum number of samples which must be in a leaf? And which splitting criterion is better?
3. What makes Random Forest classifiers more accurate and with better AUC values? Is it the splitting criterion (gini or entropy)? Is it the size of the forest? Which criterion is better and what is the best size for the forest?
4. What makes Multi-Layer Perceptron classifiers more accurate and with better AUC values? Assuming we have two hidden layers, what is the best combination of number of neurons in each layer?
5. Is there a reasonable difference between the results in terms of accuracy and in terms of AUC?
6. Which model (chosen from KNN, Decision Tree, Random Forest, MLP, SGD and Gaussian Naive Bayes) is better for this task usually on average?
7. Which model is the best for specific devices? Is it the same as the models which performed well on average?

**Second Stage: Multi-Class Classifiers**

In this stage, we will create a classifier which will be able to get a session or a sequence of sessions and determine from which device this traffic was originated. This task is much more complex than creating the One-Vs-Rest classifiers of the first stage. We will use One-Vs-Rest classifiers for each device such that the first classifier which identifies a given traffic as its own traffic, will determine the classification. In addition, we will use also the built-in multi class classifiers of the sklearn library in Python. Specifically, we will use the OneVsRestClassifier and we will check whether it performs well for different models and also comparing to our approach.

Note that there is another common approach of creating OneVsOne classifiers for each pairs of devices, but this approach has a disadvantage of a long training time. In this work we will focus on the OneVsRest approach.

# **Evaluation**

**The Dataset**

The dataset used for this research was basically collected from 10 different IOT devices: baby monitor, lights, motion sensor, security camera, smoke detector, socket, thermostat, TV, watch and water sensor. It was divided beforehand into three different datasets: train set, validation set and a test set.

The dataset contains information about the network traffic of these devices as collected for a long period of time. Each instance (example) in the dataset represents a session of connection (a TCP connection from the SYN packet to the FIN packet). The dependent variable is the classification of the device according to its type.

The training set contains approximately 400,000 instances and almost 300 features.

**Dataset Preprocessing:**

**Handling Missing Data**

It is worth mentioning that not all the data was available for all the given sessions in the dataset. It is quite common to encounter a dataset in which not all the data is available and can be used for the training. There are different approaches of how to deal with this missing data and the approach we took is removing instances with missing data. We noticed that there are still enough instances in order to make the learning effective and that the number of instances with missing data is relatively small. Note that missing data in the original dataset is represented with a question mark. When we reached the stage of testing and using the test set, we also had to deal with the fact that the “thermostat” device has only instances of one class in this test set. Hence, we were able to calculate its accuracy score, but not AUC.

**Feature Scaling**

From a quick look at the data, we noticed that the different features have different range of values. It is well-known that such changes in the features’ ranges might lead to less accurate results and problems with the training. Hence, we have decided to use the built-in MinMaxScaler of the sklearn library in Python. This scaler can be used in order to perform min-max scaling which will lead to the state that every value in the dataset is in the range (0,1). We actually noticed that after performing the feature scaling, the test set results were much accurate and with a higher AUC value.

**Feature Selection**

One of the aspects which should be considered in a lot of machine learning problems is feature selection. The concept of “the curse of dimensionality” is well-known and might cause the model to overfit or perform poorly. In order to deal with it, the feature selection concept was introduced. Some models do not need usually feature selection like Decision Trees and Random Forests. The reason is that the feature selection process is being done on-the-fly due to the way these models are being trained (the “best” feature is selected at each split of the tree).

However, some models may need feature selection to be performed in order to reach better results. In this work, it is notable that the samples-features ratio is really high (there are approximately 400,000 instances in the training set, and approximately 300 features), hence the “curse of dimensionality” should not has much impact.

**Experiments Design**

**One Vs The Rest Experiments:**

Experiment #1: Effect of Splitting Criterion in Decision Tree Classifiers

In this experiment, we would like to understand which splitting criterion in the decision trees model can lead to better results for our data: the entropy or gini criterion. The entropy criterion refers to the well-known information gain criterion which is an indication to the gain in information we have following a split in the tree. On the other hand, the Gini is a well-known criterion and indicate about the level of impurity in a specific node in the tree.

The experiment was made of the following steps:

1. Training several Decision Tree classifiers with the following values and combinations (training was done on the given training set):
   1. Splitting criterion can be: entropy or gini
   2. Minimal number of samples in a leaf: 50, 100, 200
   3. For each device
   4. 60 models were trained
2. Testing the AUC and accuracy values on the test set for each model
3. Calculating the average of the AUC for each combination of the form (splitting criterion, number of samples in a leaf)
4. According to the AUC values, we will be able to compare models which were trained with the same number of minimal samples in a leaf, but different splitting criterion. Hence, we will be able to deduce about which criterion may lead to better AUC value.

Experiment #2: Effect of The Amount of Samples in a Leaf in Decision Tree Classifiers

In this experiment, we would like to understand what is the amount of minimal samples needed in a leaf in the decision tree in order to get better results in terms of accuracy and AUC values.

The experiment was made of the following steps:

1. Training several Decision Tree classifiers with the following values and combinations (training was done on the given training set):
   1. Splitting criterion can be: entropy or gini
   2. Minimal number of samples in a leaf: 50, 100, 200
   3. For each device
   4. 60 models were trained
2. Testing the AUC and accuracy values on the test set for each model
3. Calculating the average of the AUC for each number of minimal samples in a leaf.
4. According to the AUC values, we will be able to compare the different number of minimal samples in a leaf and understand what should be this number in order to get better results for decision trees.

Experiment #3: Effect of Splitting Criterion in Random Forest Classifiers

This experiment is similar to experiment #1, but will be used in order to verify that the splitting criterion is also the best for the case of random forests (as a collection of decision trees).

The experiment was made of the following steps:

1. Training several Random Forest classifiers with the following values and combinations (training was done on the given training set):
   1. Splitting criterion can be: entropy or gini
   2. Size of the forest: 3,7,11,15,19,21,23
   3. For each device
   4. 140 models were trained
2. Testing the AUC and accuracy values on the test set for each model
3. Calculating the average of the AUC for each combination of the form (splitting criterion, forest size).
4. According to the AUC values, we will be able to compare models which were trained with the same forest size, but different splitting criterion. Hence, we will be able to deduce about which criterion may lead to a better AUC value.

Experiment #4: Effect of the Forest Size in Random Forest Classifiers

This experiment will be used in order to determine what is the recommended size of a random forest (which means, the amount of decision trees included).

The experiment was made of the following steps:

1. Training several Random Forest classifiers with the following values and combinations (training was done on the given training set):
   1. Splitting criterion can be: entropy or gini
   2. Size of the forest: 3,7,11,15,19,21,23
   3. For each device
   4. 140 models were trained
2. Testing the AUC and accuracy values on the test set for each model
3. Calculating the average of the AUC for each size of the forest
4. According to the AUC values, we will be able to compare and understand what is the recommended size of forest when using this model for our data. It will assist in the future for cases where you would like to use this model in order to choose the right size.

Experiment #5: Effect of the Amount of Neurons in Each Layer in Multilayer Perceptron Classifiers

This experiment will be used in order to understand what is the best combination of number of neurons in a two-layer perceptron.

The experiment was made of the following steps:

1. Training several Two-Layer Perceptron classifiers with the following values and combinations (training was done on the given training set):
   1. First layer will include the following amount of neurons: 1,2,3,4,5
   2. Second layer will include the following amount of neurons: 1,2,3,4,5
   3. For each device
   4. 250 models were trained
2. Testing the AUC and accuracy values on the test set for each model
3. Calculating the average of the AUC for each size of the forest
4. According to the AUC values, we will be able to compare and understand what is the recommended combination of two-layers perceptron classifiers for this kind of data

Experiment #6: Comparison Between Different Models

This experiment will be used in order to compare the average AUC values for different models trained and deducing which model is more recommended for out task

The experiment was made of the following steps:

1. Training the following classifiers: Gaussian Naive Bayes, KNN, Decision Trees, Random Forests, Multi-Layer Perceptrons
2. Calculating average AUC value for these models
3. Comparing the AUC value and checking which models lead to the best result

Experiment #7: Best Model Per Device

This experiment will be used in order to find what is the best model for each device by running all the models we trained. This experiment may give us an indication whether it is possible to have a “rule of thumb” about choosing the model for the general problem of IOT devices, or whether each device might need his own model to get good results.

The experiment was made of the following steps:

1. For each device:
   1. Run all the models trained for this device
   2. Keep track of the model with the highest AUC score
   3. Write this model as the best model for this device and its AUC score
   4. In the case of the thermostat, the comparison was made based on the accuracy score
2. According to the results, we were able to determine which model may lead to best results per device. In addition, we will be able to understand whether there is a general rule-of-thumb when choosing a model for the general problem, or whether it depends on the behaviour of the specific device.

**Multi-Class Experiments:**

Experiment #1: Accuracy Comparison of Multi-Class Classifiers Created with The Built-In OneVsRest

In the first experiment for a multi-class classifier we will try to examine the results of a classifier created with the built-in OneVsRestClassifier of the sklearn library in Python. This classifier works basically the same as our classifier in that it creates a classifier for each device (whether it is the device or not) and then decides which device is it according to the probability estimated by the different classifiers.

The experiment was made of the following steps:

1. Training OneVsRest classifiers where the internal classifier will be different at each run:
   1. Decision Tree
   2. Random Forest
   3. Linear SVC
   4. SGD (Stochastic Gradient Descent)
   5. MLP
2. We will calculate the accuracy of each classifier on the test set (ignoring instances where the label is “unknown”). Since it is a multi-class classifier, the accuracy metric should be a good indication (it is not an imbalanced data as it was in the case of the single-device classifiers).

**Evaluation Methods**

The evaluation methods were described for each experiment in the previous section. However, it is worth mentioning that in general we used the training set for the training session of the different classifiers on the data, and that we used the given test set in order to estimate the results of the classifiers.

In general, two metrics were calculated and compared in the different experiments. The first metric was the one of accuracy, while the other is the AUC (Area-Under-Curve) value. The accuracy metric represents the ratio of correct classifications of the classifier and the total number of classifications. This metric is a good indication in general, but might be problematic in case of an imbalanced dataset. In the case of the One-Vs-Rest classifiers, the data is usually imbalanced (there are much more instances for sessions which were not originated from the device), hence accuracy metric may not be enough. This is why we decided to use the AUC (Area-Under-Curve) value as well. The ROC curve represents the ratio between the true positive rate (TPR) and the false positive rate (FPR) and the closer the area under this curve is two 1 (which is the AUC), the result is better. This metric represents also how good the results are in fact of imbalanced data and focuses not only on the accuracy.

# **Results**

**One Vs The Rest Experiments:**

**Decision Trees Experiments:**

Experiment #1: Effect of Splitting Criterion in Decision Tree Classifiers

The **AUC** values for each device and algorithm combination

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Device\Algorithm | (Gini, 50) | (Gini, 100) | (Gini, 200) | (Entropy, 50) | (Entropy, 100) | (Entropy, 200) |
| Baby monitor | 0.9123 | 0.88 | 0.956 | 0.912 | 0.908 | 0.848 |
| lights | 0.625 | 0.625 | 0.625 | 0.608 | 0.608 | 0.608 |
| Motion sensor | 0.966 | 0.970 | 0.970 | 0.8738 | 0.9716 | 0.977 |
| Security camera | 0.844 | 0.845 | 0.843 | 0.845 | 0.844 | 0.852 |
| Smoke detector | 0.72 | 0.72 | 0.794 | 0.98 | 0.98 | 0.98 |
| socket | 0.691 | 0.691 | 0.691 | 0.719 | 0.719 | 0.719 |
| thermostat | Nan (accuracy 0.941) | Nan (accuracy 0.955) | Nan (accuracy 0.93) | Nan (accuracy 1) | Nan (accuracy 1) | Nan (accuracy 1) |
| TV | 0.858 | 0.852 | 0.847 | 0.828 | 0.836 | 0.835 |
| watch | 0.853 | 0.802 | 0.777 | 0.963 | 0.97 | 0.923 |
| Water sensor | 0.971 | 0.956 | 0.5 | 1.0 | 1.0 | 0.5 |

The average **accuracy** values for each model were:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (Gini, 50) | (Gini, 100) | (Gini, 200) | (Entropy, 50) | (Entropy, 100) | (Entropy, 200) |
| 0.9162 | 0.9019 | 0.8951 | 0.9235 | 0.928 | 0.9169 |

The average **AUC** values for each model were:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| (Gini, 50) | (Gini, 100) | (Gini, 200) | (Entropy, 50) | (Entropy, 100) | (Entropy, 200) |
| 0.826 | 0.816 | 0.778 | 0.858 | 0.871 | 0.804 |

Experiment #2: Effect of The Amount of Samples in a Leaf in Decision Tree Classifiers

The average **accuracy** values for each amount of samples in a leaf:

|  |  |  |
| --- | --- | --- |
| 50 | 100 | 200 |
| 0.919 | 0.9145 | 0.905 |

The average **AUC** values for each amount of samples in a leaf:

|  |  |  |
| --- | --- | --- |
| 50 | 100 | 200 |
| 0.842 | 0.843 | 0.791 |

**Random Forest Experiments:**

Experiment #3: Effect of Splitting Criterion in Random Forest Classifiers

|  |  |  |
| --- | --- | --- |
| (Criterion, Forest Size) | Average Accuracy | Average AUC |
| (gini,3) | 0.911 | 0.708 |
| (gini,7) | 0.904 | 0.688 |
| (gini,11) | 0.917 | 0.775 |
| (gini,15) | 0.928 | 0.746 |
| (gini,19) | 0.908 | 0.696 |
| (gini,21) | 0.912 | 0.754 |
| (gini,23) | 0.925 | 0.776 |
| (entropy,3) | 0.931 | 0.792 |
| (entropy,7) | 0.904 | 0.691 |
| (entropy,11) | 0.930 | 0.727 |
| (entropy, 15) | 0.914 | 0.721 |
| (entropy,19) | 0.935 | 0.782 |
| (entropy, 21) | 0.937 | 0.740 |
| (entropy, 23) | 0.931 | 0.764 |

Experiment #4: Effect of the Forest Size in Random Forest Classifiers

|  |  |  |
| --- | --- | --- |
| Forest Size | Average Accuracy | Average AUC |
| 3 | 0.921 | 0.750 |
| 7 | 0.904 | 0.690 |
| 11 | 0.923 | 0.751 |
| 15 | 0.921 | 0.733 |
| 19 | 0.921 | 0.739 |
| 21 | 0.924 | 0.747 |
| 23 | 0.928 | 0.770 |

**Multi-Layer Perceptrons Experiments:**

Experiment #5: Effect of the Amount of Neurons in Each Layer in Multi-Layers Perceptron Classifiers

|  |  |  |
| --- | --- | --- |
| Neurons Number In Each Layer | Average Accuracy | Average AUC |
| (1,1) | 0.910 | 0.5 |
| (1,2) | 0.877 | 0.594 |
| (1,3) | 0.885 | 0.777 |
| (1,4) | 0.887 | 0.775 |
| (1,5) | 0.886 | 0.592 |
| (2,1) | 0.910 | 0.5 |
| (2,2) | 0.901 | 0.742 |
| (2,3) | 0.910 | 0.5 |
| (2,4) | 0.871 | 0.675 |
| (2,5) | 0.902 | 0.775 |
| (3,1) | 0.886 | 0.652 |
| (3,2) | 0.891 | 0.775 |
| (3,3) | 0.878 | 0.777 |
| (3,4) | 0.919 | 0.780 |
| (3,5) | 0.912 | 0.787 |
| (4,1) | 0.904 | 0.782 |
| (4,2) | 0.909 | 0.726 |
| (4,3) | 0.919 | 0.782 |
| (4,4) | 0.891 | 0.780 |
| (4,5) | 0.903 | 0.782 |
| (5,1) | 0.880 | 0.628 |
| (5,2) | 0.919 | 0.780 |
| (5,3) | 0.897 | 0.770 |
| (5,4) | 0.895 | 0.779 |
| (5,5) | 0.8953 | 0.776 |

**General Experiments:**

Experiment #6: Comparison Between Different Models

Average AUC comparison between the different models (ignoring the thermostat)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| KNN | Decision Tree | Random Forest | MLP | SGD | Gaussian Naive Bayes |
| 0.849 | 0.802 | 0.740 | 0.711 | 0.873 | 0.534 |

Experiment #7: Best Model Per Device

|  |  |  |
| --- | --- | --- |
| Device | Model | Score (AUC) |
| Baby Monitor | KNN (K=5) | 1 |
| Lights | MLP (5,2) | 0.9375 |
| Motion Sensor | MLP(5,2) | 0.995 |
| Security Camera | MLP(3,4) | 0.99875 |
| Smoke Detector | Random Forest (Gini, 7) | 1 |
| Socket | KNN (K=5) | 0.9375 |
| TV | MLP (5,5) | 0.97375 |
| Watch | Decision Tree (Entropy, 100) | 0.969375 |
| Water Sensor | Decision Tree (Entropy, 100) | 1 |
| Thermostat | Decision Tree (Entropy, 100) | 1 (Accuracy) |

**Multi-Class Experiments:**

Experiment #1: Accuracy Comparison of Multi-Class Classifiers Created with The Built-In OneVsRest (single session)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Decision Tree | Random Forest | MLP | SGD | Linear SVC |
| 0.754 | 0.631 | 0.6 | 0.636 | 0.718 |

# **Discussion**

**One Vs The Rest:**

**Model Selection:**

One of our research questions in the above-mentioned experiments was what is the suggested model to use for the One-Vs-Rest task for a specific device. During the experiments and specifically in experiment #6 it seems that the best AUC value is being achieved using the SGD (Stochastic Gradient Descent) classifier. It is worth mentioning that the AUC value was chosen as a good indication since it relies on the False Positive Rate and the True Positive Rate unlike the accuracy metric which might mislead in case of imbalanced data.

In previous works (like the one relating to ProfilIOT), the main model was the random forest. Although the average AUC value (for all the different devices) is still quite good (0.74), it is still less than other models we tried for this task. In experiment #3, we can also see there is a big difference between the accuracy values and the AUC values for the decision trees. It might imply that this model is good for the cases where accuracy is more important, but it doesn’t mean it will achieve high score if FPR and TPR are both important.

The SGD, KNN (K-Nearest Neighbours) and Decision Trees seems to have better AUC value (on average). The fact that SGD got the best score may imply that the data may be linearly-separable and when approaching such task of IOT traffic classification, it may be a good model to start working with.

As mentioned, KNN had high AUC value as well (0.849 on average), but the training time was much longer than the one for the SGD. This can be a good reason to choose SGD instead of KNN in case we are limited with training time available.

**Difference Between Metrics:**

From all the experiments in this stage, it is quite notable that there is a difference between the accuracy metric and the AUC metric. Most of the models got high accuracy values, although it is not the same for the AUC metric. Since in a lot of tasks from the kind of One-Vs-Rest we have an imbalanced data, we must consider what is the metric which is more relevant for us. A better indication will probably be the AUC since it relates to both FPR and TPR, unlike accuracy which in case of imbalanced data can lead to high score.

**Minimal Number of Samples In a Leaf Should Be Low in Decision Trees:**

As expected, the minimal number of samples in a leaf in the decision trees should be low. The more samples which can be in a leaf, the less the tree will trained and splitted into new nodes. Note that it is still recommended to avoid overfitting of the data by setting this parameter to a really low number. In our case, 100 samples in a leaf led to the highest AUC value of 0.843 which is quite well and did not require a lot of training time (much less than KNN for example).

**Effect of Random Forest Size on Results:**

It is quite clear from experiment #4, that the more decision trees included in the random forest, the better AUC value was achieved. The maximal size of the forest which tested was 23 trees which led to the result of 0.77.

In future work, it may be considered that maybe additional trees can lead to better results. Note that the amount of trees must be odd since we use majority vote.

**Gini and Entropy as Splitting Criterion in Decision Trees:**

As can be seen from both experiments #1 and #3, the entropy splitting criterion usually leads to better results (especially in terms of AUC, but also in terms of accuracy). If we take a look at experiment #1 we can see that for the same number of samples in a leaf but with different splitting criterion we get different results, where entropy seems to work better.

In the sklearn library of Python, entropy relates to the splitting criterion of Information Gain. According to this criterion, the chosen feature for the next split in the decision tree is chosen based on the maximal decrease in entropy following the split. On the other hand, the Gini criterion is another criterion for impurity. It is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

According to the results, it seems that in most cases for a data like we have in this research, the information gain criterion may be more useful.

**Number of Neurons in Two-Layers Perceptron:**

According to the results in experiment #5, the combination of neurons which led to the best AUC value is of 3 neurons in the first layer and 5 neurons in the second layer (with AUC of 0.787). It may be a good indication of what number of neurons it is good to start when working on such data.

In general, it seems that the AUC values are quite similar as we add more and more neurons in the different layers. The worst result was when we had only one neuron in each layer (result of 0.5, similar to a random classifier).

The results may indicate that it is best to have more neurons in each layer, although these results are not decisive (for example, a neural network of 2 neurons in each layer led to an AUC value of 0.742, while a neural network of (2,3) led to 0.5.

In future work, it may be a good idea to experiment the number of layers in neural networks and its effect on the performance and results.

**Model Selection According to the Device:**

Is is quite notable from experiment #7 that each device may need different model in order to get its best results. While the data distribution for some device may be better with the K-Nearest Neighbours model, others perform better with the Random Forests. The conclusion is that although we noticed some models lead to better results on average, it is still better to choose the model at the level of a specific device. The most common models we see in experiment #7 as the best models are decision trees and Multi-Layer Perceptrons. However, the models still might differ with their relevant parameters (for different device for example, different number of neurons in each layer is required).

**Multi-Class Classification:**

**SKlearn OneVsRestClassifier Model Selection (at the Level of a Single Session):**

In experiment #1, we can see that there is hugh importance to the selection of the basic model to be used in the OneVsRestClassifier of sklearn. As mentioned, this classifier creates behind-the-scenes 10 different classifiers for each device and then chooses the general device classification according to the probabilities of certainty in the classification. It is part of the sklearn library in Python and simply requires a basic classifier instance as its input.

It is quite notable from the results that in such case (where all internal classifiers are being created from the same model), the Decision Tree classifier reaches the highest accuracy score (of 0.754). Another classifier which performs quite well is the linear SVC and the rest perform poorly.

These results may indicate that decision tree model and the linear SVC are more recommended for this task of multi-class classification at the level of a single session.

# **Appendix**

1. ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis - Yair Meidan , Michael Bohadana , Asaf Shabtai , Juan David Guarnizo , Mart´ın Ochoa , Nils Ole Tippenhauer , and Yuval Elovici (2017)

2. Detection of Unauthorized IoT Devices Using Machine Learning Techniques - Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan David Guarnizo, Yuval Elovici (2018)

3. Unknown Malware Detection Using Network Traffic Classification - Dmitri Bekerman, Bracha Shapira, Lior Rockach, Ariel Bar (2015)