RIT

Rochester Institute of Technology

CSEC 520/620: Cyber Analytics & Machine Learning

Assignment 4 - IoT Classification

Due Date - November 14, 2021 11:59 PM

Purpose

The purpose of this assignment is to learn how decision trees and random forests can be used to classify Internet-of-Things devices based on their network traffic. In this assignment, your main task is to implement the decision tree and random forest algorithms to identify types of IoT devices based on network traffic.

Description

You and your team will implement

- Decision Tree (DT) and
- Random Forest (RF)

You may use external libraries (e.g. scikit-learn) so long as you are not using the libraries' implementation of DT/RF classification (or any key steps such as Gini impurity computation).

Dataset and Initial Code

Download the dataset and initial code from here. This is a processed slice of the dataset published by Sivanathan et al. in *Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics*.

- Their work described an ensemble classifier composed of a Multinomial Naive Bayes classifier and a Random Forest to identify devices using network traffic in two stages. A general overview of the system can be seen in Figure 1.
- We recommend you read their paper to better understand the classification system you will be working on.
- In this assignment, we've provided you some base code to work with. The provided code is a re-implementation of the work as described by Sivanathan et al. Your goal is to replace the scikit-learn Random Forest classifier with your own custom-built version.

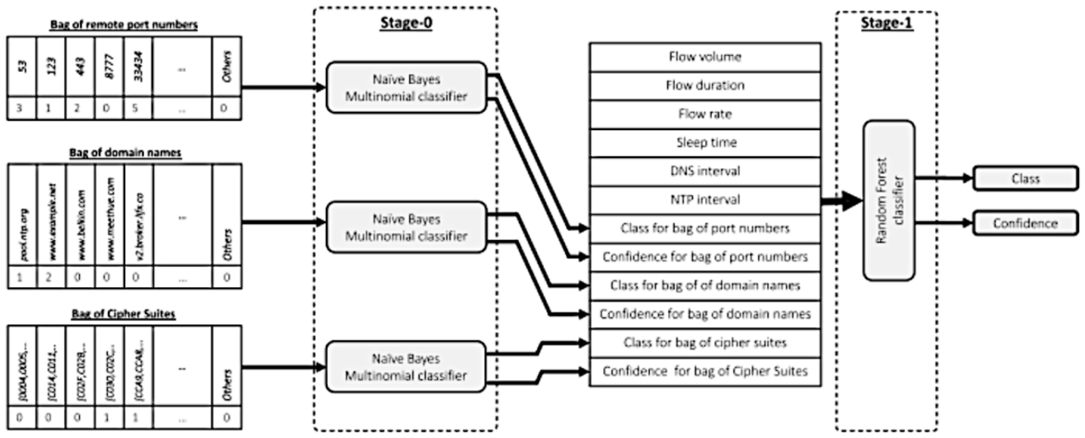

Figure 1: IoT Device Classification Process (copied from Sivanathan et al.)

Steps

1. **Working with the code**
    - Install the packages specified in the requirements.txt file in a Python 3 environment.
      `pip install -r requirements.txt`
    - Run the code with arguments to predict using the scikit learn library.
      `python classify.py -r ./iot_data`
    - Your objective is to replace the code in the `do_stage_1(...)` function with your own code that will perform the Random Forests classification.

2. **Algorithms**

A Random Forest is an ensemble of multiple Decision Trees, so you will first need to code the Decision Tree algorithm before you can use it in a Random Forest classifier.

(2.1) Decision Tree

- Define the following parameters:
    - The maximum depth of the tree, *max_depth*.
    - The minimum number of samples allowed per leaf node, *min_node*.
- Write a function to compute Gini impurity given data that has been split into two groups.
- Write a function that uses Gini impurity to find the best location to split your samples (e.g. lowest impurity) by checking each feature point in your samples to be split.
- To build your decision tree, keep splitting the dataset and resulting splitted groups until each split reaches one of the following conditions:
    - The branch has reached the *max_depth*.
    - The number of samples in the group to split is less than the *min_node*.
    - The optimal split results in a group with no samples, in which case use the parent node.
    - The optimal split has a worse Gini impurity than the parent node, in which case use the parent node.

- To predict, navigate the tree using your test sample until you reach a leaf node. The predicted class is the mode of the sample labels in that node.

(2.2) Random Forest

- Define the following parameters:
  - The number of decision trees, *n_trees*, to use for your forest.
  - The percentage of the dataset, *data_frac*, to use when building each tree.
  - The number of features, *feature_subcount*, to sample when performing splits during tree building.
- Build *n_trees* trees using following process:
  - Before building the tree, build a subset of the data at *data_frac* size by random sampling *with replacement* from training samples.
  - Next, the tree can be built normally with one modification: When evaluating the Gini impurity of each test split, use only a subset of the features to calculate the impurity. Like you did when you built the data subset, randomly sample features *with replacement* until you have *feature_subcount* features, and then calculate the impurity.
- Predict by voting: To evaluate a test sample, run it through each tree to gather class 'votes'. The class with the most votes is the final classification for the test sample.

3. Performance metrics

The multiclass accuracy your model achieves against the IoT device dataset.

Accuracy = C / (C + W)

C = Correct prediction of the correct network device for the sample,
W = Incorrectly predicted a different network device for the sample.

4. Hyperparameter Tuning

Explore the parameters of both the algorithms for hyperparameter tuning.

Deliverables

1. All source code of your implementations.
   - <u>Document your code!</u> Your code documentation should demonstrate you understand what your code is doing (and that you did not just copy-and-paste from an external source).
2. A readme file that should contain:.
   - A clear description of the directions to set up and run your code.
   - If your code requires external libraries, or if not written in Python, provide the additional references/direction.

   If your instructions are not clear, your work will ***<u>NOT</u>*** *be graded!*
3. A short report describing your experiments and the results of your evaluations (~3 pages).

<u>Expected Report Elements:</u>

1. Hyperparameter tuning
   a. Explain your hyperparameter tuning process.
   b. Identify which parameters had the largest impact on performance for your

Random Forest model. Use plots to visualize your results, but **avoid using too many and over-explaining**.

2. Performance
    a. Evaluate the performance of your Decision Tree model (non-ensemble) to your Random Forest model against the data.
        i. How do their execution time and performance compare?
        ii. Explain why you think you get these results.
    b. Include a normalized confusion matrix of your results to demonstrate the accuracy of your model on a class-by-class basis.
    c. Discuss the results in your confusion matrix.
    d. Are there any classes that are commonly confused with one another? Why or why not may that be the case? (**Hint:** Use the *list_of_devices.txt* to translate MAC address labels into device types to help explain the classifier confusion).
3. How useful would it be to normalize the data? Explain your answer.
4. Select one non-trivial node (i.e. not a leaf) in one decision tree in your final forest. Print out all the information used at that node to compute Gini impurity. Show how to use this information to compute the node importance for that node, and do the computation. See for details: https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3
5. Include any citations in an appropriate and consistent format.
6. ***Not Expected*:** Background on the DT, RF, the paper, or the dataset. Especially if it leads you to copy other material (see "A Note on Plagiarism" below!).

Grading Rubric

Criteria	1 Poor	2 Basic	3 Proficient	4 Distinguished
Tree and Forest Implementation (35%)	No or non-functioning code for anomaly identification techniques.	Code for is mostly functional, but parts of the implementation have significant errors.	Minor errors or significant inefficiencies exist in implementation, but are otherwise mostly correct.	Techniques are perfectly correct. Code is cleanly written and handles most edge cases.
Experiments & Results (20%)	Minimal results are reported.	Results are incomplete and/or wrong, or are not presented in a way where its meaning is clear.	A reasonable set of results showing expected performance. Some presentation issues such as confusing graphs/tables or unnecessary detail.	The classifier achieves the expected performance and all experiments are presented clearly.
Analysis & Questions (20%)	No or nonsensical analysis. Missing most questions.	Analysis is inaccurate or hard to understand. Some, but not all, questions are addressed.	Analysis is sensible and covers questions well. May be missing appropriate references or is weak in some areas.	Clear and accurate analysis that is backed up with appropriate references.
Writing Quality (15%)	Very poorly written and hard to follow.	Major points are visible, but writing may include many errors and/or lack focus and is disorganized.	Writing is clear enough to be understood, but some points may lack focus. Relatively few writing errors.	The paper is clear and well organized. Writing is smooth and polished with very few errors.
Code Documentation (10%)	Inadequate documentation provided both in the code and with the code.	Acceptable documentation for running the code, but lacking in-code documentation and helpful descriptions to demonstrate understanding.	Code includes some documentation (e.g. function docstrings, inline comments), however quality may be weak or unclear.	High quality documentation is provided. The purpose and function of all segments of code can easily be understood.

A: 3.25 average or higher
B: 2.5 - 3.25 average.
C: 1.75 - 2.5 average.

A Note on Plagiarism

When writing, you must include in-line citations whenever you are reporting on information that is not “general knowledge,” i.e. anything you learned for this project and didn’t know in advance. This is **NOT** just for quoted information. Failure to do this is plagiarism.

This article on plagiarism is good and covers the line between common knowledge and other material: https://writingcenter.unc.edu/tips-and-tools/plagiarism/. Also: https://www.plagiarism.org/ has a ton of additional information.

I have had students fail to follow these guidelines and get caught nearly every time I’ve taught my research seminar. These students get put on probation and have even been suspended from the university for this serious academic violation. *Please do not be the next!*