CSC 7210: Project 1

Katherine E. Brown

Introduction

Modern intrusion detection is reliant upon machine learning (Khraisat et al. 2019). Particularly, anomaly-based intrusion detection system relies on machine learning to differentiate attacks from normal activity based on their inherent features rather than known signatures. Decision trees are machine learning techniques that essentially derives if-then rules based on what conditions differentiate classes best. This report details the development of a decision tree based on a modified version of the KDD Cup 1999 dataset and compares this tree against other decision tree implementations.

Exploratory Data Analysis

In this section, I will discuss the dataset and exploratory data analysis.

Dataset Information

The dataset used in this experiment is a modified version of the KDD Cup 1999 dataset (Eberle 2020; KDD). The original dataset (KDD) detailed records for 24 known attacks in its training set with 14 present in the test data. Each of these known attacks can be classified as one of the following: denial of service, user to root, or network probe. The original dataset contains a total of 41 features. Of these 41 features, 34 are continuous and 7 are categorical. These features describe the TCP or UDP connections themselves, detected actions and conditions once connected, and traffic information of each connection. Further information regarding the original dataset can be found (KDD).

The modified dataset used to construct the Decision Tree contains a binary classification task. This task is to classify connections as either a "Neptune" attack or as benign (Eberle 2020). The "Neptune" attack is a SYN-flood denial of service attack (KDD ; Eberle 2020).

The modified dataset consists of 11 features and 800 records from the original dataset. Features that were continuous in the original dataset were discretized based on their values. Table 1 contains a listing of each feature and their possible values.

Table 1: Table detailing modified dataset features and possible values

Feature Name	Possible Values
duration	zero one+ ten+ hundred+ thousand+ tenThousand+
protocol_type	tcp udp icmp
count	zero one+ ten+ hundred+ thousand+ tenThousand+
srv_count	zero one+ ten+ hundred+ thousand+ tenThousand+
serror_rate	zero 1-24 25-49 50-74 75-99 oneHundred
srv_serror_rate	zero 1-24 25-49 50-74 75-99 oneHundred
rerror_rate zero	1-24 25-49 50-74 75-99 oneHundred
srv_rerror_rate	zero 1-24 25-49 50-74 75-99 oneHundred
same_srv_rate	zero 1-24 25-49 50-74 75-99 oneHundred
diff_srv_rate	zero 1-24 25-49 50-74 75-99 oneHundred
srv_diff_host_rate	zero 1-24 25-49 50-74 75-99 oneHundred

The classes in the modified training set are evenly split with the Neptune class containing 400 instances and the normal class also containing 400 instances. Of the 11 included features, rerror_rate and srv_rerror_rate consist of the same value. Also, the duration, protocol_type, diff_srv_rate, and srv_diff_host_rate each had over 99% majority of one value. The remaining features, count, srv_count, serror_rate, srv_serror_rate and same_srv_rate, each have nontrivial distributions.

The testing data consists of 200 records, and much like the training data, the Neptune and normal classes each have 100 values. The features duration, protocol_type, rerror_rate, srv_error_rate, and diff_srv_rate each have only value across all the instances. The remaining features contain nontrivial distributions.

Dataset Challenges

There are several features in the dataset that essentially allow the classifier to "cheat" at the classification task. These features are given in Table 2 for the training set and Table 3 for the testing set.

Further, there are features that, while their values do not necessarily form a mapping to the resulting class, are highly indicative of the class. For example, in both the training and testing sets, when the value is "hundred+" for count directly classifies to the Neptune class.

Table 2: Table that list "cheating" features in the training set.

Feature	Values when Normal	Values when Neptune
serror_rate	zero 25-49	75-99 oneHundred
srv_serror_rate	zero 25-49	oneHundred

Table 3: Table that list "cheating" features in the testing set.

Feature	Values when Normal	Values when Neptune
serror_rate	zero	75-99
srv_serror_rate	zero	oneHundred
same_srv_rate	oneHundred	zero 1-24

These findings in the dataset lead to a hypothesis that trees formed on the complete dataset will be near perfectly accurate. Thus, selecting feature subsets based on the above analysis will help provide a clearer evaluation of the implemented decision tree.

Development of Decision Tree

In this section, I will discuss the development of a decision tree in Python.

Development Environment

The custom decision tree is implemented using Python 3.7.4 (Van Rossum and Drake 2009). This language was chosen for its simplicity and portability across different systems. Further, it contains built-in data structures and access to data processing and manipulation and numerical computing tools.

To manage and process the data efficiently, I used Pandas version 0.25.1 (McKinney 2010). For efficient numerical calculations, I used NumPy version 1.17.2 (Harris 2020).

Development Experience

Development of the decision tree was aided with the use of Python and Russell and Norvig's Artificial Intelligence textbook (Russell and Norvig 2009). Python has many built-in data structures that can be used in a variety of ways to accomplish tasks. A design choice for this decision tree was the use of Python dictionaries to represent the decision tree. Figure 1 shows the decision tree for the classic restaurant dataset detailed in (Russell and Norvig 2009). To read the tree, at the current indentation level, the "attribute_name" key represents which attribute is providing the split-point, and the remaining keys are values of attributes that link to their specific action.

The tree is implemented using pseudocode provided (Eberle 2020) and pseudocode and discussion in (Russell and Norvig 2009). Attributes chosen to provide the split-point at a specific level of the tree are done so using information gain. Russell and Norvig provide pseudocode for Iterative Dichotomiser 3 (ID3) algorithm.

Overall, the provided pseudocode and references (Eberle 2020; Russell and Norvig 2009), along with development design choices, made the development of the decision tree more efficient and easier to debug. However, due to the previous discussion of the dataset, I opted to use the restaurant dataset described in (Russell and Norvig 2009). Since the algorithm is implemented according to the textbook, I was able to verify my decision tree implementation by comparing my tree to those within the textbook.

```
attribute_name: pat
some : yes
full :
        attribute_name: hun
        yes :
                attribute_name: type
                French : normal
                Thai :
                        attribute_name: fri
                        yes : yes
                        no : no
                Burger : yes
                Italian : no
        no : no
none : no
```

Figure 1: Decision Tree on Restaurant Dataset

Table 4: Listing of the feature subsets evaluated and their included features.

Subset Name	Included Features
Subset 1	duration protocol_type count srv_count rerror_rate srv_rerror_rate diff_srv_rate srv_diff_host_rate
Subset 2	duration protocol_type srv_count rerror_rate srv_rerror_rate diff_srv_rate srv_diff_host_rate

Experimentation

In this section, I discuss experimentation.

Experiment Methodology

The first experiment performed is evaluation of each decision tree upon the full dataset with all features. The goal of this experiment is to evaluate the Custom decision tree with a well-known implementation of the CART decision tree algorithm from Scikit-Learn (Pedregosa 2011). A comparison of the CART and ID3 algorithms can be found at (Singh and Gupta 2014).

Based on the exploratory data analysis, the decision trees will also be evaluated on two feature subsets to evaluate the decision trees independent of overly advantageous features. Table 4 lists the names and features of these subsets for further reference.

Overview of Decision Trees Evaluated

For both experiments, the custom decision tree developed in this report is compared to implementations of the CART algorithm. For reference, the decision trees evaluated are summarized in Table 5 by their algorithm, source of implementation, feature selection criterion, and splitting strategy.

Results

The results of the experiment on the entire dataset is given in Table 6. As shown in Table 6, both decision trees yield perfect training and testing accuracy.

Table 5: Table depicting the decision trees evaluated in experiments 1 and 2

Name	Algorithm	Source of Implementation	Split Criterion
Custom	ID3	Custom	Entropy
SK-Entropy	CART	Sci-Kit Learn	Entropy

Table 6: Accuracy of each tree on the full dataset

Tree	Training Accuracy (in %)	Testing Accuracy (in %)
Custom	100%	100%
SK-Entropy	100%	100%

The training and testing accuracy on Subset 1 is within 7% and 4% of the training and testing accuracy entire feature space, respectively.

Table 7: Accuracy of each tree on Subset 1

Tree	Training Accuracy (in %)	Testing Accuracy (in %)
Custom	93.5%	96.5%
SK-Entropy	93.75%	96.5%

Finally, Subset 2 yields a major drop in both training and testing accuracy compared to the entire feature set and Subset 1.

Table 8: Accuracy of each tree on Subset 2

Tree	Training Accuracy (in %)	Testing Accuracy (in %)
Custom	64.125%	76.5%
SK-Entropy	64.25%	76.5%

Discussion

In this section, I will discuss three important observations from the results of Experiment 1.

Effect of Feature Removal

First, as hypothesized, both the custom decision tree (Custom) and the SciKit-Learn decision tree (SK-Entropy) both were able to achieve impossibly high accuracy on the training and testing data. This is undoubtedly a result of the highly indicative features in the dataset.

Subset 1 removes features that were identified in the exploratory data analysis as providing a clear split between Normal and Neptune classes. When these features are removed, accuracy on both decision trees degrades; however, it is still above 90%. Thus, we can conclude that the features removed to form Subset 1 provided an unfair advantage to the classifier.

Although accuracy above 90% is welcomed on classification tasks, we remove one more feature that could be creating an unfair advantage in Subset 2. Accuracy degrades substantially between Subset 1 and Subset 2 for both evaluated decision trees. This implies that the "count" feature was providing enough information to help the classifiers obtain high classification accuracy.

Comparison of CART and Custom Decision Tree

Generally, both the custom decision tree and the SciKit-Learn decision tree performed within 1% of each other. When accuracy differed between the two trees, the SciKit-Learn was the more accurate decision tree. This is most likely a result of the CART algorithm's performance gains over ID3 (Singh and Gupta 2014). Nonetheless, however, we can conclude the custom decision tree is comparable to other decision tree implementations.

Training and Testing Data Trend

Finally, we note an interesting trend in the training and testing accuracy. It is usually expected that the testing accuracy be less than the training data, since testing data is unseen by the classifier. For Subset 1 and Subset 2, however, the testing data is higher than the training data. This is likely a result of more features in the testing set taking on same values. Future work could evaluate on a real-world dataset to evaluate whether the decision tree models are truly generalizing from the data or overfitting.

Conclusion

This report detailed the development of a decision tree algorithm to detect network intrusions. Exploratory data analysis revealed potentially compromising features in the dataset. The developed decision tree was compared to known implementations from a popular Python machine learning library, SciKit-Learn, and found to be comparable.

References

Eberle, W. 2020. Project assignment: Intrusion detection. *CSC 7210 - Anomaly and Intrusion Detection*.

Harris, Charles R., e. a. 2020. Array programming with NumPy. *Nature* 585:357–362.

Kdd cup 1999 data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Accessed: 2020-10-09.

Khraisat, A.; Gondal, I.; Vamplew, P.; and Kamruzzaman, J. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2(1):20.

McKinney, W. 2010. Data Structures for Statistical Computing in Python. In Stéfan van der Walt, and Jarrod Millman., eds., *Proceedings of the 9th Python in Science Conference*, 56 – 61.

Pedregosa, F., e. a. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. USA: Prentice Hall Press, 3rd edition.

Singh, S., and Gupta, P. 2014. Comparative study id3, cart and c4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST)* 27(27):97–103.

Van Rossum, G., and Drake, F. L. 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.