Machine Learning for Enterprise Cybersecurity

Fall 2019

Goals

- Introduce basics of machine learning (ML) for cybersecurity
- Introduce a few useful open-source ML tools

Outline

- What is supervised machine learning?
- Motivating example: detecting malicious commands (e.g., Bash, PowerShell)
  - Problem definition
  - How to detect using heuristics
  - How to detect using machine learning
- Potential pitfalls when using supervised machine learning
- Other forms of machine learning and data science for cybersecurity

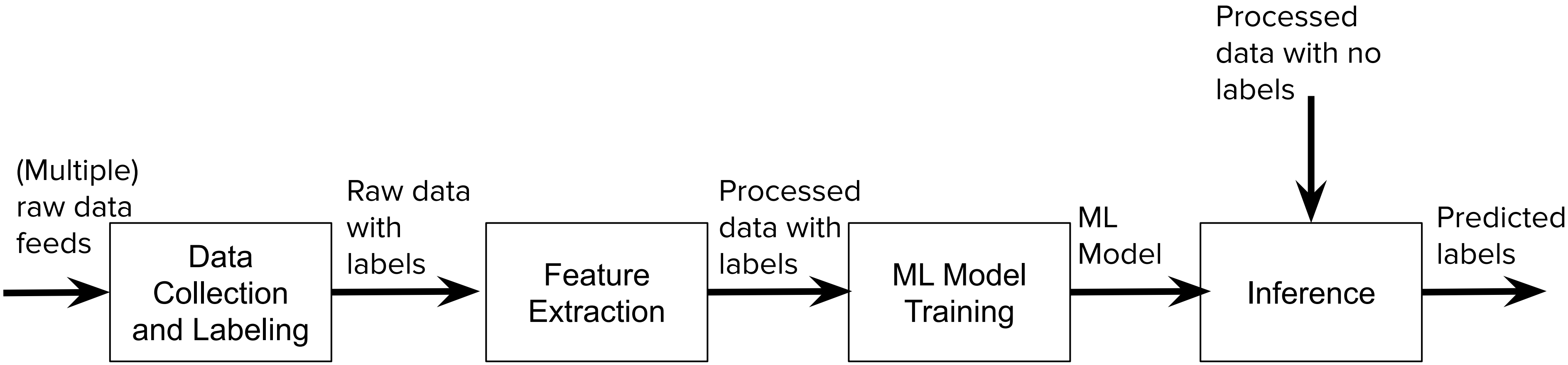
Supervised machine learning: classification

- Class of algorithms that automatically learns a method to distinguish datapoints into pre-defined categories
- Widely used for many problems
- For cybersecurity, use cases include differentiating:
    - Malware from normal files
    - Malicious SQL queries from normal SQL queries
    - Types of traffic
    - Many others

Supervised machine learning pipeline

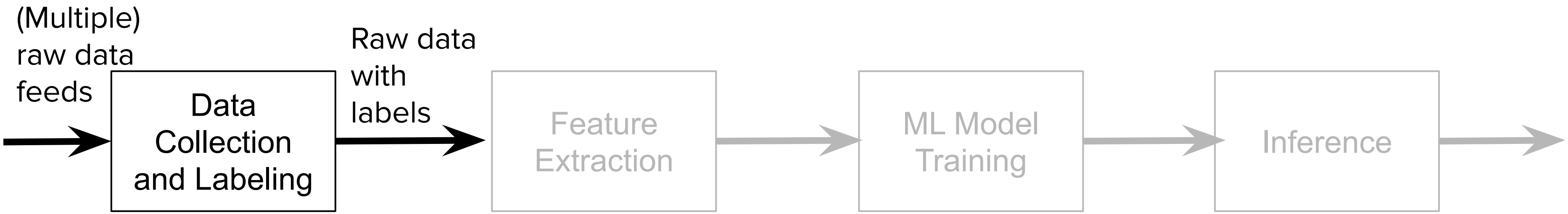
- A typical supervised ML pipeline can be (roughly) divided into four stages:
  - Data collection and labeling
  - Feature extraction
  - ML model training
  - Inference
- Note: This is not the only way to conceptually divide up a supervised ML pipeline!

Supervised machine learning pipeline



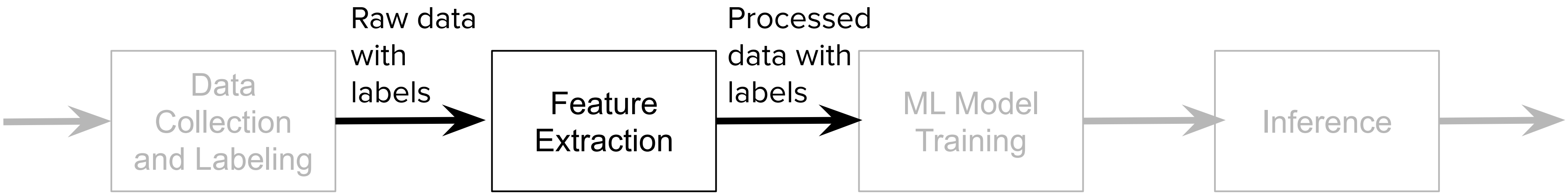
Supervised ML pipeline: data collection and labeling

- Data collection and labeling stage
  - Collect the raw data that will be used in later stages
  - Creates labels



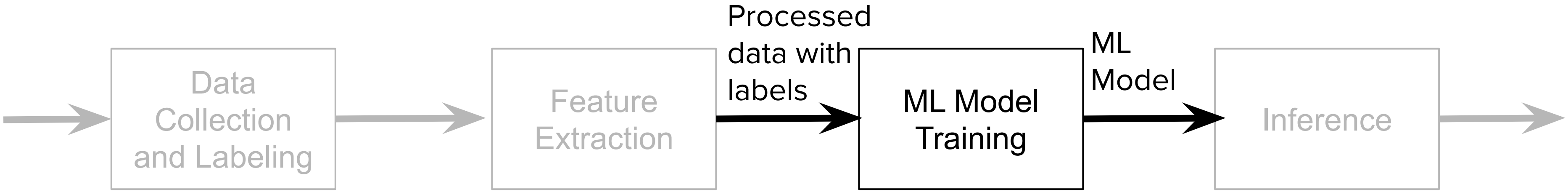
Supervised ML pipeline: feature extraction

- Feature extraction stage
    - Converts raw data into a format that the ML model can understand
    - Often, goal is essentially to create a large table of numerical values



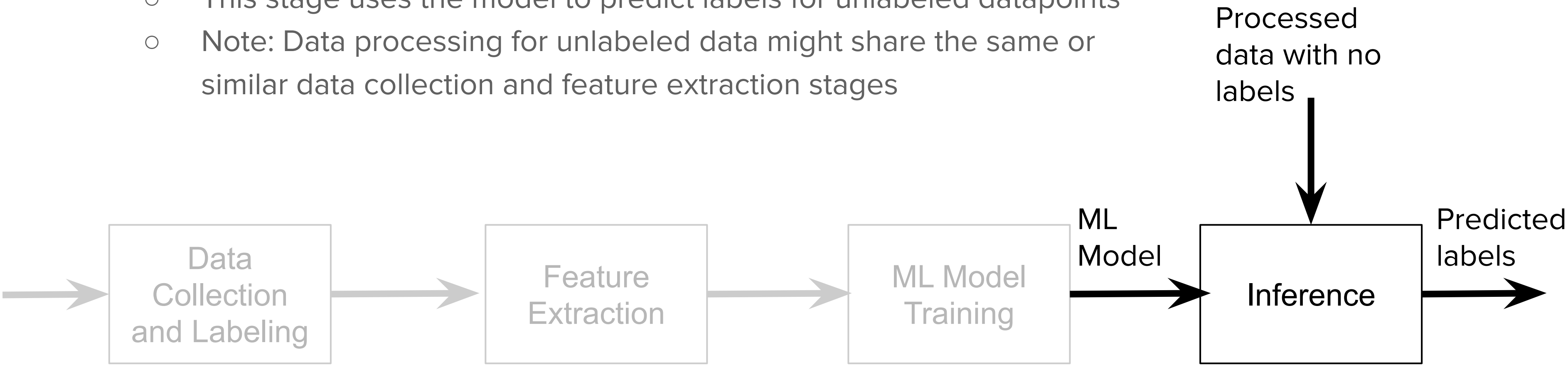
Supervised ML pipeline: ML model training

- ML model training stage
  - Uses labeled data from previous stage to create a ML model
  - ML model is capable of predicting labels for unlabeled datapoints



Supervised ML pipeline: inference

- Inference stage
  - This stage uses the model to predict labels for unlabeled datapoints
  - Note: Data processing for unlabeled data might share the same or similar data collection and feature extraction stages



Malicious Bash/Powershell commands

- Once in a system, these tools are natively available
- Attackers can use them for many parts of the attack lifecycle
- This is an example of “living off the land”

Heuristics for detecting malicious Bash/PowerShell

- A very simple solution is to create a list of static signatures
- Example: If any commands match ‘cat /etc/passwd’, raise an alert
  - That is, use exact string matches
- Useful but incomplete security solution
  - Will not catch anything that does not match exactly
  - Adversaries will purposely try to avoid commands that match signatures

Obfuscated Bash/PowerShell

- To avoid signatures, attackers can purposely obfuscate their commands
- Scripts exist to automatically obfuscate Bash/Powershell commands
  - Bashfuscator: https://github.com/Bashfuscator/Bashfuscator
  - Powershell Invoke-Obfuscation: https://github.com/danielbohannon/Invoke-Obfuscation

Obfuscated command detection

- Pretty easy for humans to differentiate between obfuscated and non-obfuscated commands
- Can we build a detector that can do this automatically for us?
- Why use this as a motivating example
  - Easy to tell difference as a human
  - Easy to determine what the detectors are doing
  - General techniques towards building a detector are widely applicable
  - Will learn all building blocks of a detector that could be deployed in the real world

Baseline detector: manually-encoded decision tree

- Let’s manually create some baseline detectors for obfuscated bash
- Rough characteristics of obfuscated bash compared to unobfuscated bash:
  - Much more punctuation
  - Tends to be longer
  - Unusual sequences of characters
  - Certain methods of obfuscation tend to always insert the same words
- We can try to encode our domain knowledge as a tree of if-then rules

Machine learning version of baseline decision tree

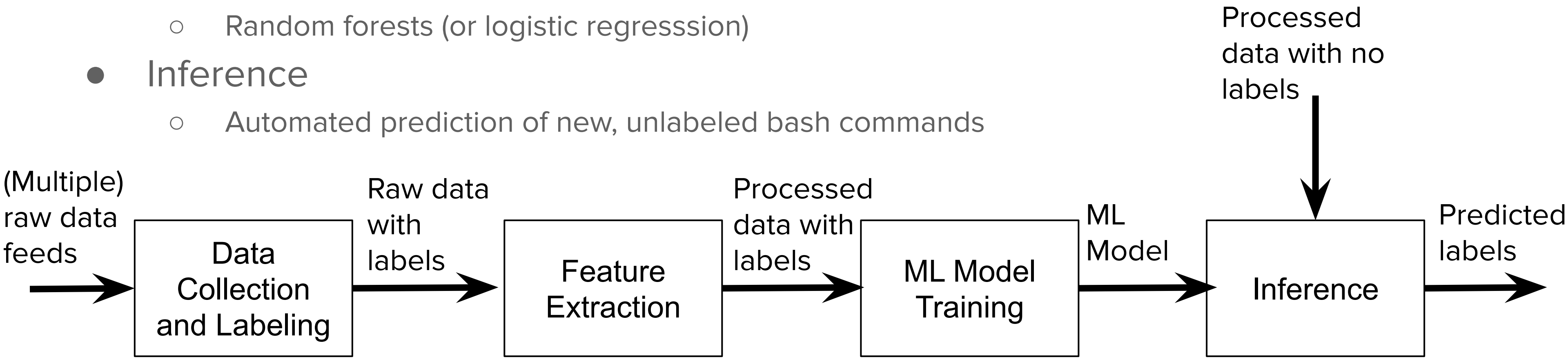
- Major problems with manually creating a decision tree:
  - What criteria do we use at each tree node?
  - What order should we place the nodes?
  - At what value should we make each split?
  - How many nodes should be in the tree?
- Machine learning versions of decision trees exist that automatically make these decisions based on available data

Random forests

- Instead of creating a single tree, we can create many trees (a forest of trees) and take a majority vote
- To create a diverse set of trees, each tree in the forest is created using a random subset of the data
- This is the intuition behind “random forests”
- This model works well in practice and is a useful baseline approach

Supervised ML pipeline for obfuscated bash

- Goal: Classify obfuscated versus unobfuscated bash commands
- Data collection and labeling
  - Collection of unobfuscated bash; corresponding output from bashfuscator
- Feature extraction
  - Unusual characters, command length, character sequences
- ML model training
  - Random forests (or logistic regresssion)
- Inference
  - Automated prediction of new, unlabeled bash commands



Another baseline detector: a linear model

- Rough characteristics of obfuscated bash compared to unobfuscated bash:
  - Much more punctuation
  - Tends to be longer
  - **Unusual sequences of characters**
  - **Certain methods of obfuscation tend to always insert the same words**
- Let’s build another detector that looks for unusual sequences of characters
- This model will basically accumulate evidence that the command is obfuscated or not

Another baseline detector: mathematically expressed version

- We can pose this new baseline in a number of mathematical formats

Data-driven version of baseline detector

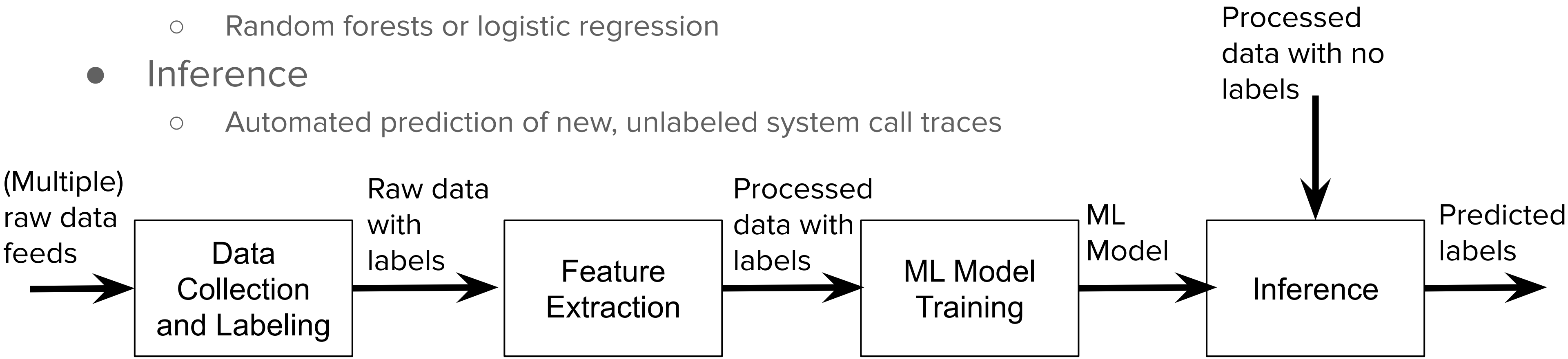
- By automatically learning the weights in the various mathematical formulations of our baseline, we essentially have a linear machine learning model
- This is another useful class of supervised classification models
    - One particularly useful linear model is logistic regression

Supervised learning for system call data

- Small modifications to previous pipeline allow us to handle automated detection of malicious system call sequences
- Roughly, all we really need to change are:
  - Data collection
  - Feature extraction
- The flexibility of ML approaches for many different types of problems is one of the most powerful aspects of ML

Supervised ML pipeline for system call traces

- Goal: Classify system call sequence as normal versus malicious
- Data collection and labeling
    - Collection of system call traces
- Feature extraction
    - System call sequences
- ML model training
    - Random forests or logistic regression
- Inference
    - Automated prediction of new, unlabeled system call traces



Supervised learning for system call data

- Simple feature extraction
  - Ignore system call parameters
  - Use sliding window of n system calls (an n-gram of system calls)
  - Datapoint is a normalized histogram of system call n-grams
- Features can then be used to learn a model (e.g., via logistic regression)
- Simple but straightforward baseline

Supervised classification for cybersecurity

- Method to automatically place datapoints into classes (i.e., categories)
- Possible cybersecurity use cases:
  - Normal vs. malicious
  - Different types of normal behavior
- Pros:
  - Powerful set of techniques
  - Potentially useful for a variety of use cases
- Cons:
  - Requires labeled data (potentially a lot of it)
  - Human interpretation not always straightforward

Issues with supervised ML for cybersecurity

- For supervised learning to work, typically assume:
    - We have (lots of) labeled data
    - Training data reflects data during inference
- Cybersecurity tends to break both of these assumptions

Issues with supervised ML for cybersecurity: dataset issues

- Data is unavailable
  - Labeling data is (very) time consuming/difficult
  - Data cannot be shared (e.g., due to data privacy)
  - Attacks tend to be rare events, so hard to get lots of examples

Issues with supervised ML for cybersecurity: dataset issues

- Data is not reflective of real world
  - Ratio of normal/malicious data is not reflective of real life
  - Data is artificially generated and contains generation artifacts
  - Normal and malicious data come from different sources and contain source artifacts

Issues with supervised ML for cybersecurity: concept drift

- Training data typically does not fully reflect data during inference because:
  - Adversaries will purposely evade existing detectors
  - New attacks will occur during inference that do not occur during training
  - Frequency of certain types of attacks changes over time
- Unfortunate reality:
  - Often very difficult to gauge robustness of a machine learning classifier
  - Machine learning classifiers must be periodically re-trained, otherwise performance drops over time
    - How to automatically detect when they should be re-trained is an open issue
    - Different groups have different ways of doing this

Interaction between evaluation metrics and ratio of normal to malicious data

- When testing ML on cybersecurity, need to make sure the evaluation metric and the ratio of normal to malicious data matches reality
- Unless careful, very bad detectors can appear to do well
  - Consider a classifier that marks everything as normal
- In the following slides, we will look at this effect on some example metrics

Example evaluation metric: TP, TN, FP, FN

- In binary classification problems, convention is to call the two classes the "positive class" and "negative class"
- There are four types of predictions:
  - ***True Positive (TP)***: Correctly predicting the positive class
  - ***True Negative (TN)***: Correctly predicting the negative class
  - ***False Negative (FN)***: Incorrectly predicting a positive class member to be in the negative class
  - ***False Positive (FP)***: Incorrectly predicting a negative class member to be in the positive class

Example evaluation metric: accuracy

- Accuracy = $\dfrac{TP + TN}{TP + TN + FP + FN}$

- Measures what percentage of the predictions are correct

Example evaluation metric: FPR, FNR

- FPR = $\dfrac{FP}{TN + FP}$

- FNR = $\dfrac{FN}{TP + FN}$

- Measures false positive rate (FPR) and false negative rate (FNR)

Example evaluation metric: precision, recall, F1-score

- Precision = $\dfrac{TP}{TP + FP}$
- Recall (i.e., sensitivity) = $\dfrac{TP}{TP + FN}$
- F1-measure is the harmonic mean of precision and recall

Confusion matrix

		Predicted Class: **Positive**	Predicted Class: **Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	**Negative Predictive Value** $\frac{TN}{(TN + FN)}$	**Accuracy** $\frac{TP + TN}{(TP + TN + FP + FN)}$

Effect of relative and absolute class balance on evaluation metrics

- Ratio of number of normal and malicious values can affect metric values
- Absolute number of normal and malicious values can affect metric values

Other machine learning paradigms

- Reinforcement learning
- Unsupervised learning
  - Clustering
  - Anomaly detection

Unsupervised learning: clustering

- Method to group similar datapoints together in a dataset

Clustering for cybersecurity

- Method to group similar datapoints together in a dataset
- Choice of clustering algorithm and parameters dictates what is considered similar
- Possible cybersecurity use cases:
  - Grouping together different types of behavior
  - Grouping together different types of users
- Pros:
  - Does not require labels
  - Potentially useful for a variety of use cases
- Cons:
  - Clusters may or may not correspond to groupings considered useful by humans

Unsupervised learning: anomaly detection

- Method to find the most unusual datapoints in a dataset

Anomaly detection for cybersecurity

- Method to find the most unusual datapoints in a dataset
- Assumptions when applied to cybersecurity:
  - Assumes malicious events are rare
  - Assumes malicious events are unusual
  - Choice of anomaly detector dictates what is considered unusual
- Pros:
  - Does not require labels
  - Can potentially catch new types of malicious behavior
- Cons:
  - Assumptions might not be true in practice
  - False positive rate often very high in practice
  - Is not typically designed to improve over time

Isolation forests

- An anomaly detector that often performs well in practice
- Works by repeatedly and randomly partitioning the feature space
- Number of partitions needed to ‘isolate’ a data point indicates how unusual the data point is

Other forms of data-driven analysis

- Simple statistics are often sufficient for cybersecurity
- For example, simply looking at min/max values, commonly occuring values, etc., are often useful

Summary

- Machine learning is a useful tool for many problem domains including cybersecurity
- Machine learning is not a silver bullet solution for cybersecurity
    - Simple solutions are often sufficient in practice
    - Need to be careful with how algorithms are trained and deployed
- These slides barely scratch the surface in terms of what machine learning can and cannot do, as well as robust methods of creating/deploying machine learning in practice