Network Intrusion Detection

Expo Presentation

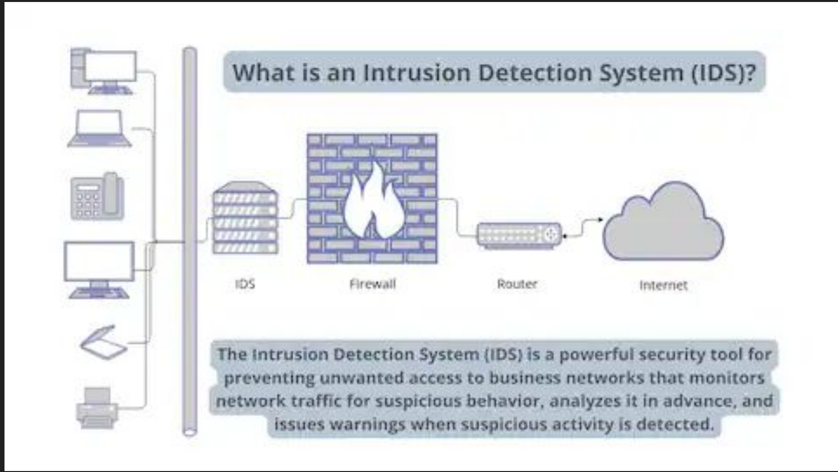
Network Intrusion Detection Project

- One problem in Network Security is detecting malicious actors trying to connect to computer networks

- By training on existing datasets of network traffic data, we can use ML classifiers to detect attackers and their intended attack

- We used the KDD Cup ‘99 Dataset, a dataset developed by DARPA for this purpose

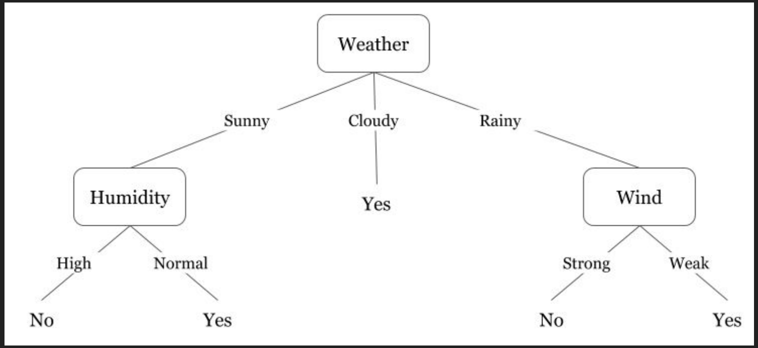
- Students learned about the basics of network security and ML classifiers to create their own intrusion detection models



Decision Tree

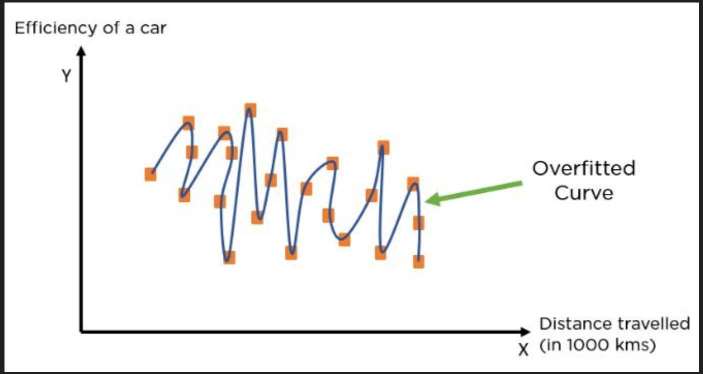
Overview:

- Recursively splits data based on most prominent feature until there is a usable map from input, down the tree to expected output.
- Very sensitive and volatile to small changes.
- Works best with classifications.



Challenges That Came With This Model:

- Large amount of overfitting (the model was performing too good to be true)
- By experimenting with the hyperparameters, we were able to lessen the model's overfitting
- Introducing artificial noise could also be another option.
- Also less random jumping from local maximas could help.



Training Procedure:

- Data was eliminated by importance to prediction
- Model was trained over many iterations to refine the "questions" for branches
- This process is done using "information gained", an information theory concept.

Performance on dataset:

- Model performance was evaluated on testing data.
- High performance, 99% correct classification
- Outliers likely coming from very little data on some attack types.

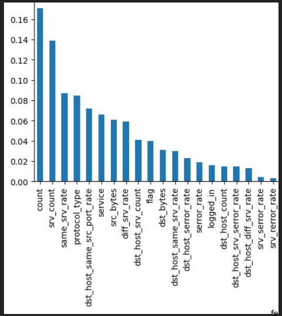
Support Vector Machine

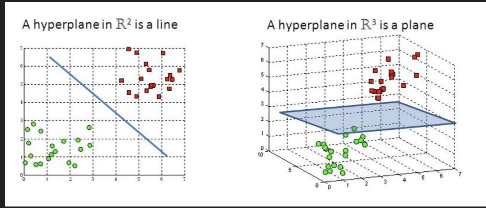
In Theory

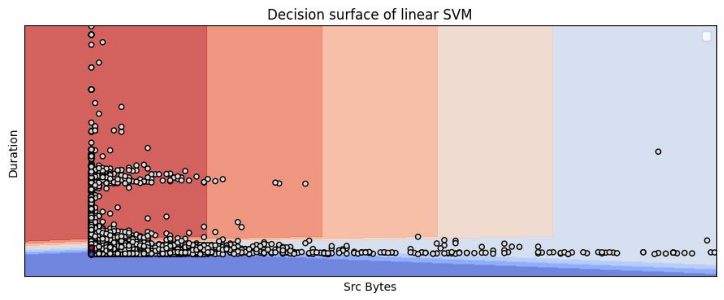
- Works similar to regression analysis, drawing a boundary line between positives and negatives
- Pros:
    - Effective in high dimensional spaces with multiple characteristics
    - Contained in many libraries that offer good results with no tuning
- Cons:
    - Easy to overfit training data
    - Can get easily confused if data is very noisy

In Practice (On our Dataset)

- Filtered data to focus on main types of attacks
- Scaled data to eliminate variance
- Encoded data to change categorical variables to numerical data
- Balanced performance and accuracy by deciding which variables to use in classification
    - Decided to use all variables to get highest accuracy without significant performance decrease
- Achieved accuracy of .9999
- Took a lot of time to get data into useable formats



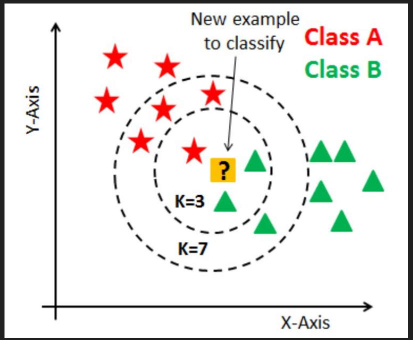





```
Confusion Matrix:
[[21253     0     0]
 [    2 19537     2]
 [    0     4 56256]]
0.999175716611372
```

K-Nearest Neighbors (KNN)

- Overview:
  - Classifies data points by looking at all data points within a certain distance *k* of the given data point, and decides it based on what the majority of the nearby data points are
  - *k* is the only hyperparameter and it can be tuned to prevent overfitting using cross-validation
  - With multiple predictors, the Euclidean distance is taken to classify the data point
- Strengths of this model:
  - No need to train the data set
    - begin sorting test data right away
  - The algorithm is simple
    - easy to implement and debug
  - Can easily adjust *k* to avoid overfitting
- Challenges/Weaknesses:
  - Computationally intensive, since all points must be stored in memory
  - Takes a long time to compute, so testing the model was time-consuming
  - Assumes that similar data points will be near each other



Model Accuracy:

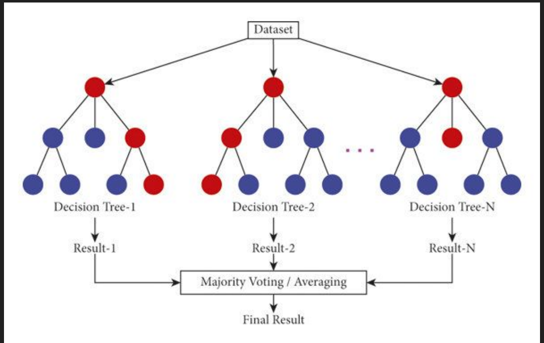
0.9993927420530176

Classification report:

	precision	recall	f1-score	support
back	0.99	0.99	0.99	569
buffer_overflow	0.75	0.67	0.71	9
ftp_write	0.00	0.00	0.00	1
guess_passwd	0.82	1.00	0.90	9
imap	0.00	0.00	0.00	2
ipsweep	0.98	0.97	0.97	307
land	1.00	1.00	1.00	6
loadmodule	0.00	0.00	0.00	1
multihop	0.67	0.67	0.67	3
neptune	1.00	1.00	1.00	26951
nmap	0.91	1.00	0.95	51
normal	1.00	1.00	1.00	24291
perl	0.00	0.00	0.00	1
phf	0.00	0.00	0.00	1
pod	1.00	0.96	0.98	56
portsweep	0.98	0.93	0.95	257
rootkit	0.00	0.00	0.00	1
satan	1.00	0.97	0.98	407
smurf	1.00	1.00	1.00	70098
teardrop	1.00	1.00	1.00	244
warezclient	0.97	0.98	0.97	237
warezmaster	0.00	0.00	0.00	4
accuracy			1.00	123506
macro avg	0.64	0.64	0.64	123506
weighted avg	1.00	1.00	1.00	123506

Random Forest

Jenny Lee



- Overview
  - bootstrap create multiple mini datasets with replacement out of the original dataset
    - *with replacement to preserve the original distribution of the data
  - add randomness by choosing a best feature out of a subset of all the features to split on
    - helps decorrelate the mini datasets
  - have multiple trees (# estimators) and take the average
    - like tossing a coin, the more trees/trials, the more representative the prediction is
  - for each bootstrapped mini dataset, make a decision tree where features are selected to split on if they decrease entropy the most
  - either take the most popular or average prediction out of all the classifiers
  - must encode strings as digits
- Cons/Challenges
  - time consuming ~3 hours
  - not very high performance - did especially poorly on 'normal' target
  - PERFORMANCE: weighted avg precision of .93 and f1-score of .90

	precision	recall	f1-score	support
0.0	0.90	0.80	0.85	23924
1.0	0.52	1.00	0.69	9757
2.0	1.00	0.90	0.95	61643
accuracy			0.88	95324
macro avg	0.81	0.90	0.83	95324
weighted avg	0.93	0.88	0.90	95324

- Training
  - CV of 5 with grid search to tune hyper parameters
  - selected from top important features calculated by sklearn package
- Pros
  - Not susceptible to overfitting
  - Familiar Logic