UNIVERSITY of WEST FLORIDA

Performance of Machine Learning Algorithms on UWF-ZeekData22

Sikha Bagui, *Dustin Mink & Subhash Bagui*

University of West Florida, USA

UNIVERSITY of WEST FLORIDA

Tactics in UWF-ZeekData22

label_tactic	count
Discovery	2086
Lateral Movement	4
Privilege Escalation	13
Reconnaissance	9278722
Persistence	1
Initial Access	1
Exfiltration	7
Defense Evasion	1
none	9281599
Resource Development	3
Credential Access	31

Malicious vs. Non-malicious traffic in UWF-ZeekData22

Non-Malicious Traffic	9,281,599
Malicious Traffic	9,280,869

Reconnaissance and Discovery Tactics

- Reconnaissance tactic
  - Carried out with the goal of gathering information, to plan future attacks
  - 9,278,722 instances
- Discovery tactic
  - Used to learn specifics of network infrastructure
  - 2086 instances

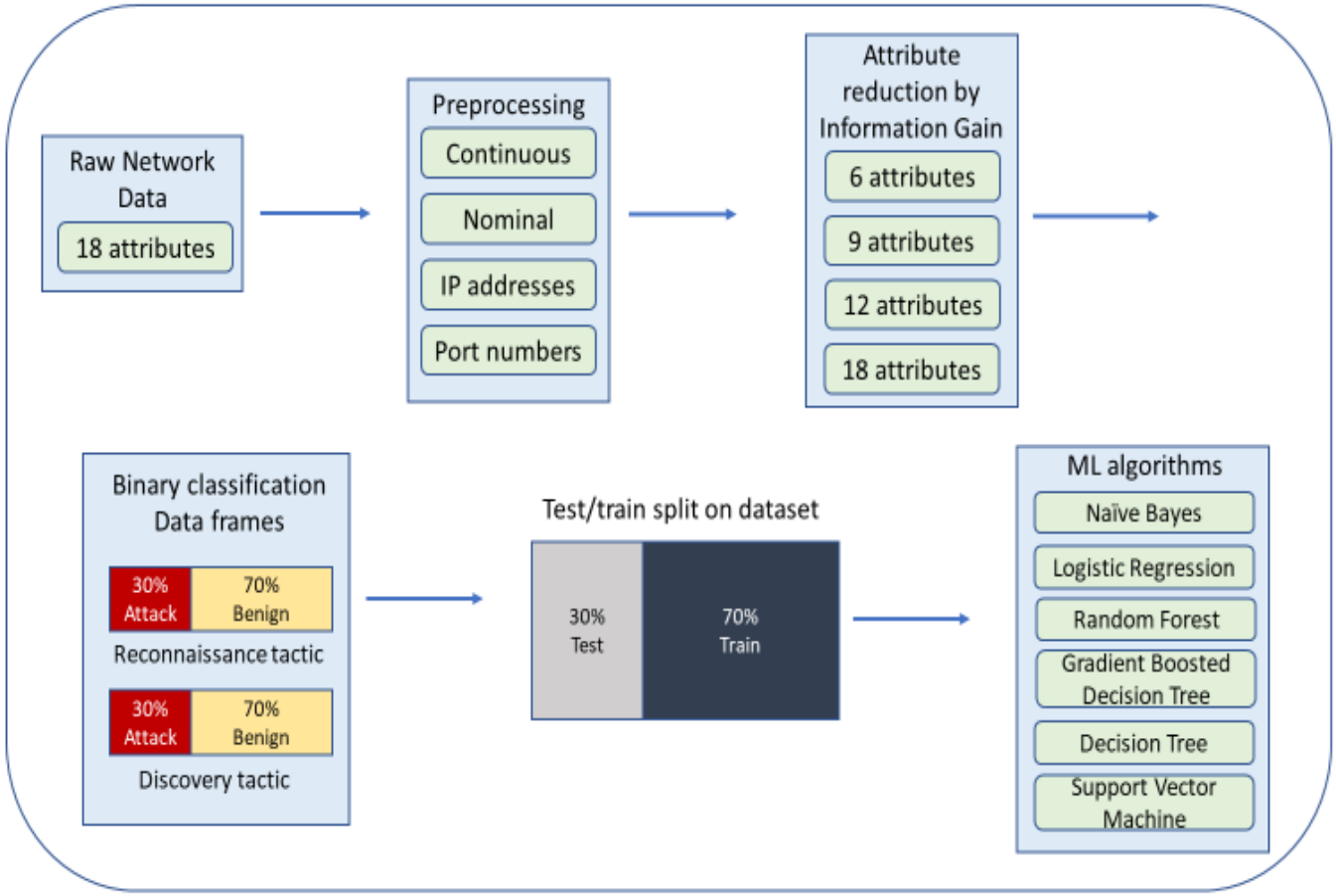
UNIVERSITY of WEST FLORIDA

Features in Zeek Conn Log File

```
>>> df_conn.printSchema()
root
 |-- resp_pkts: integer (nullable = true)
 |-- mitre_attack: string (nullable = true)
 |-- service: string (nullable = true)
 |-- orig_ip_bytes: integer (nullable = true)
 |-- local_resp: boolean (nullable = true)
 |-- missed_bytes: integer (nullable = true)
 |-- proto: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- conn_state: string (nullable = true)
 |-- dest_ip_zeek: string (nullable = true)
 |-- orig_pkts: integer (nullable = true)
 |-- community_id: string (nullable = true)
 |-- resp_ip_bytes: integer (nullable = true)
 |-- dest_port_zeek: integer (nullable = true)
 |-- orig_bytes: integer (nullable = true)
 |-- local_orig: boolean (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- history: string (nullable = true)
 |-- resp_bytes: integer (nullable = true)
 |-- uid: string (nullable = true)
 |-- src_port_zeek: integer (nullable = true)
 |-- ts: double (nullable = true)
 |-- src_ip_zeek: string (nullable = true)
```

UNIVERSITY of WEST FLORIDA

Experimentation…



UNIVERSITY of WEST FLORIDA

Preprocessing

Preprocessing Using Binning

- Zeek Conn logs contain: continuous valued attributes, nominal attributes, IP addresses and port numbers.
- Continuous Attributes
  - The trimmed mean and standard deviation was calculated for each column
  - To address the skewness caused by lengthy and low-lying tails, a 10% trim on the data was used to generate the bins

Binning IP Addresses and Port Numbers

The commonly recognized network classifications of A, B, C, D, and E were used, each of which pertain to specific ranges of the first octet in the IP address

Port Numbers were binned based on IANA rules

UNIVERSITY of WEST FLORIDA

Information Gain

- Used to assess the relevance of the 18 features in the Zeek Conn Logs

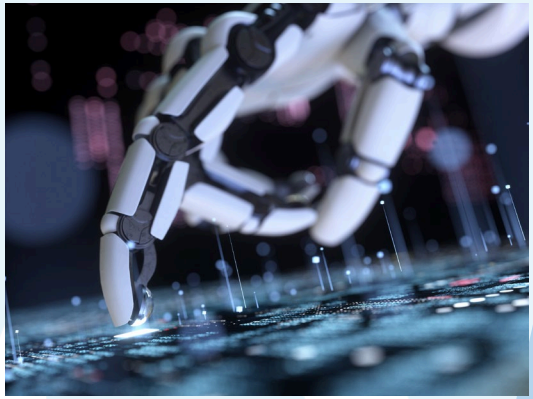
Attribute no.	attribute	info_gain
1	history	0.827
2	protocol	0.77
3	service	0.726
4	orig_bytes	0.724
5	dest_ip	0.674
6	orig_pkts	0.655
7	orig_ip_bytes	0.572
8	local_resp	0.524
9	dest_port	0.486
10	duration	0.386
11	conn_state	0.166
12	resp_pkts	0.085
13	resp_ip_bytes	0.065
14	src_port	0.008
15	resp_bytes	0.008
16	src_ip	0.007
17	local_orig	0.002
18	missed_bytes	0

UNIVERSITY of WEST FLORIDA

Computing Environment

University of West Florida's Hadoop cluster was used

- Cluster has 6 worker nodes.
- Spark 3.2.1 and Hadoop 3.3.1.



UNIVERSITY of WEST FLORIDA

Determining Spark's Optimum Parameters

UNIVERSITY of WEST FLORIDA

Testing Spark's Configuration Parameters

Configuration	Spark Property	Description
Driver Cores	spark.driver.cores	Number of cores used for the driver process, only in cluster mode.
Driver Memory	spark.driver.memory	Amount of memory used for the driver process, i.e. where SparkContext is initialized.
Executor Cores	spark.executor.cores	Number of cores used on each executor.
Executor Memory	spark.executor.memory	Amount of memory used per executor process.
Executor Instances	spark.executor.instances spark.dynamicAllocation.minExecutors	Initial number of executors to run if dynamic allocation is enabled, with upper and lower bounds for the number of executors established.
	spark.dynamicAllocation.maxExecutors	
Shuffle Partitions	spark.sql.shuffle.partitions	Default number of partitions used when shuffling data for joins or aggregations.

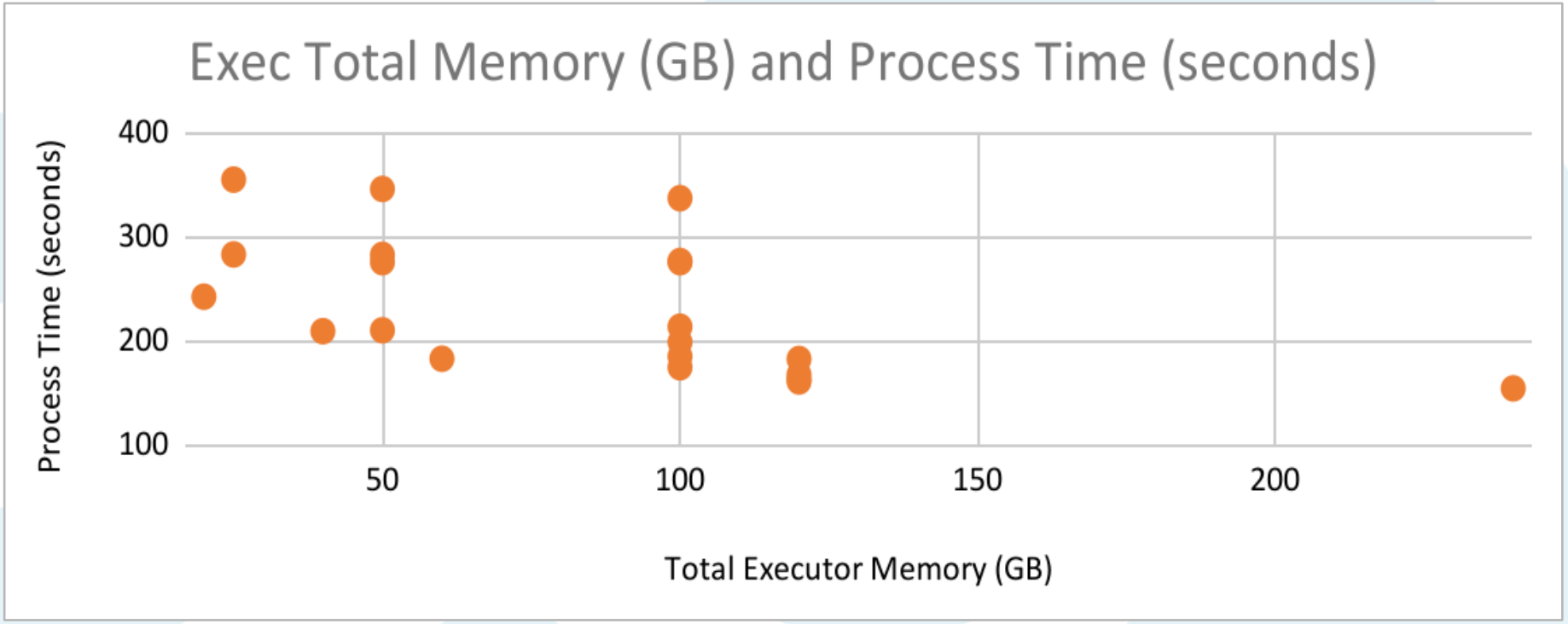
UNIVERSITY of WEST FLORIDA

Performance Results with various Spark Executor Parameters

Test ID	Executor Count	Cores Per Executor	Memory Per Executor	Total Executor Cores	Total Executor Memory (GB)	Total Time (seconds)	Shuffle partitions	Driver Cores	Driver memory (GB)
1	5	2	5	10	25	355.24	200	2	10
2	5	2	10	10	50	346.30	200	2	10
3	5	2	20	10	100	337.45	200	2	10
4	5	4	5	20	25	283.60	200	2	10
5	5	4	10	20	50	276.21	200	2	10
6	5	4	20	20	100	277.28	200	2	10
7	10	2	5	20	50	283.18	200	2	10
8	10	2	10	20	100	276.20	200	2	10
9	10	4	5	40	50	210.99	200	2	10
10	10	4	10	40	100	199.78	200	2	10
11	20	2	5	40	100	214.43	200	2	10
12	20	4	5	80	100	186.02	200	2	10
13	10	8	10	80	100	175.59	200	2	10
14	12	8	10	96	120	172.91	200	2	10
16	12	8	10	96	120	171.49	72	2	10
17	12	8	10	96	120	164.35	12	2	10
18	24	4	5	96	120	183.51	24	2	10
19	6	16	20	96	120	162.02	6	2	10
20	12	8	10	96	120	170.18	24	2	10
21	3	32	40	96	120	168.58	3	2	10
22	1	16	20	16	20	243.16	1	2	10
23	2	16	20	32	40	210.14	2	2	10
24	6	32	40	192	240	155.39	6	2	10
25	3	16	20	48	60	183.75	3	2	10
26	10	8	10	80	100	178.59	200	2	10
27	6	32	40	192	240	156.8	6	2	10
28	6	32	40	192	240	161.84	12	2	10
29	6	32	40	192	240	159.12	24	2	10
30	6	32	40	192	240	159.31	48	2	10
31	6	32	40	192	240	159.91	96	2	10
32	6	32	40	192	240	161.13	192	2	10
33	6	32	40	192	240	161.6	384	2	10
34	6	32	40	192	240	159.2	6	4	10
35	6	32	40	192	240	157.79	6	4	20
36	6	32	40	192	240	158.2	6	4	30

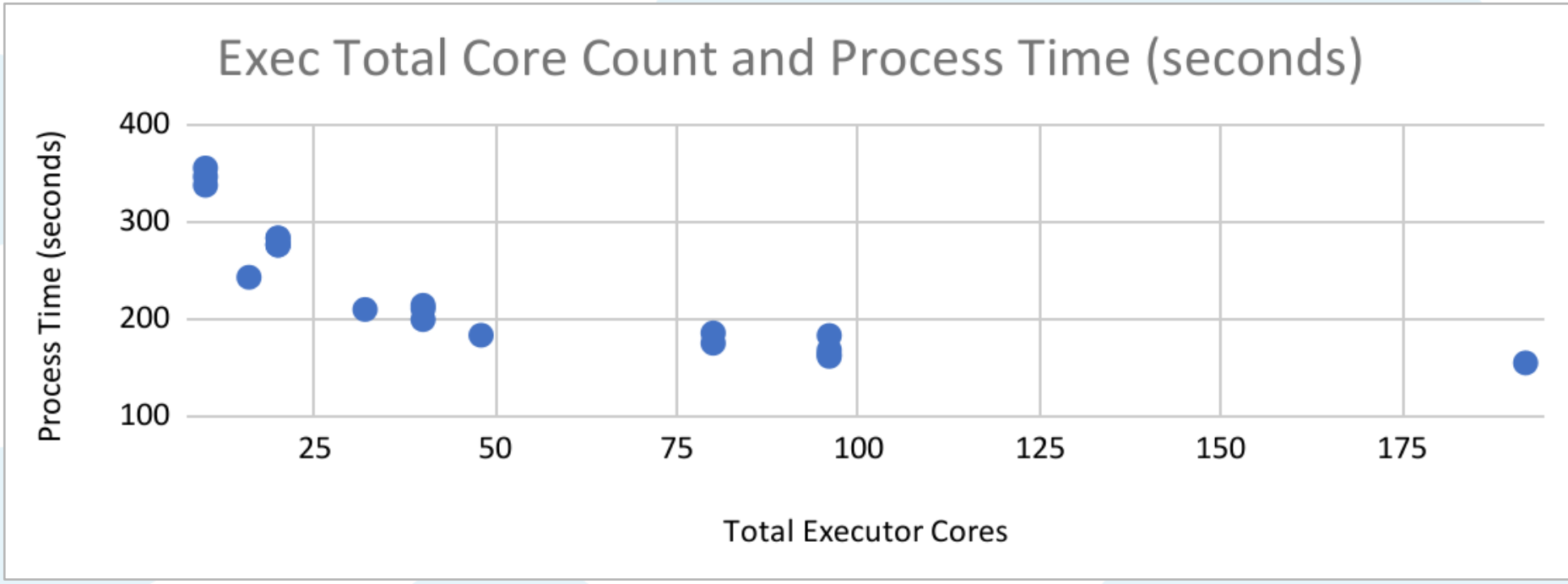
Process Times vs Memory Per Executor

- Executor memory -- amount of memory (in GBs) assigned to each executor. There does **not** appear to be a strong correlation between the total executor memory and processing time



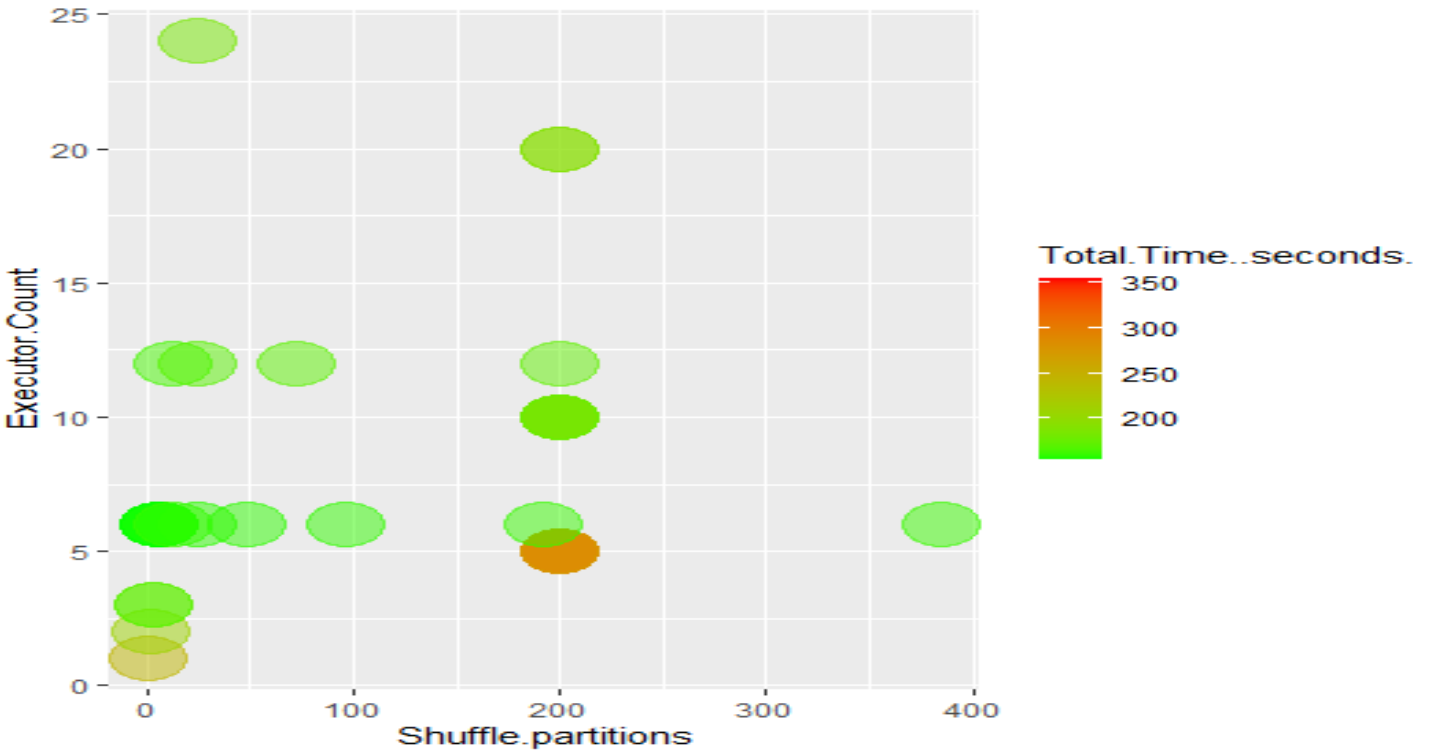
Process Times vs Number of Executor Cores

- Increasing the number of cores available from 10 to 50 had a significant impact on performance, with diminishing returns apparent after a total core count of 100.



Processing Times as a Function of Executor Count and Shuffle partitions

- 6 and 12 executors show that using fewer shuffle partitions improves performance



UNIVERSITY of WEST FLORIDA

Spark's best Config Settings

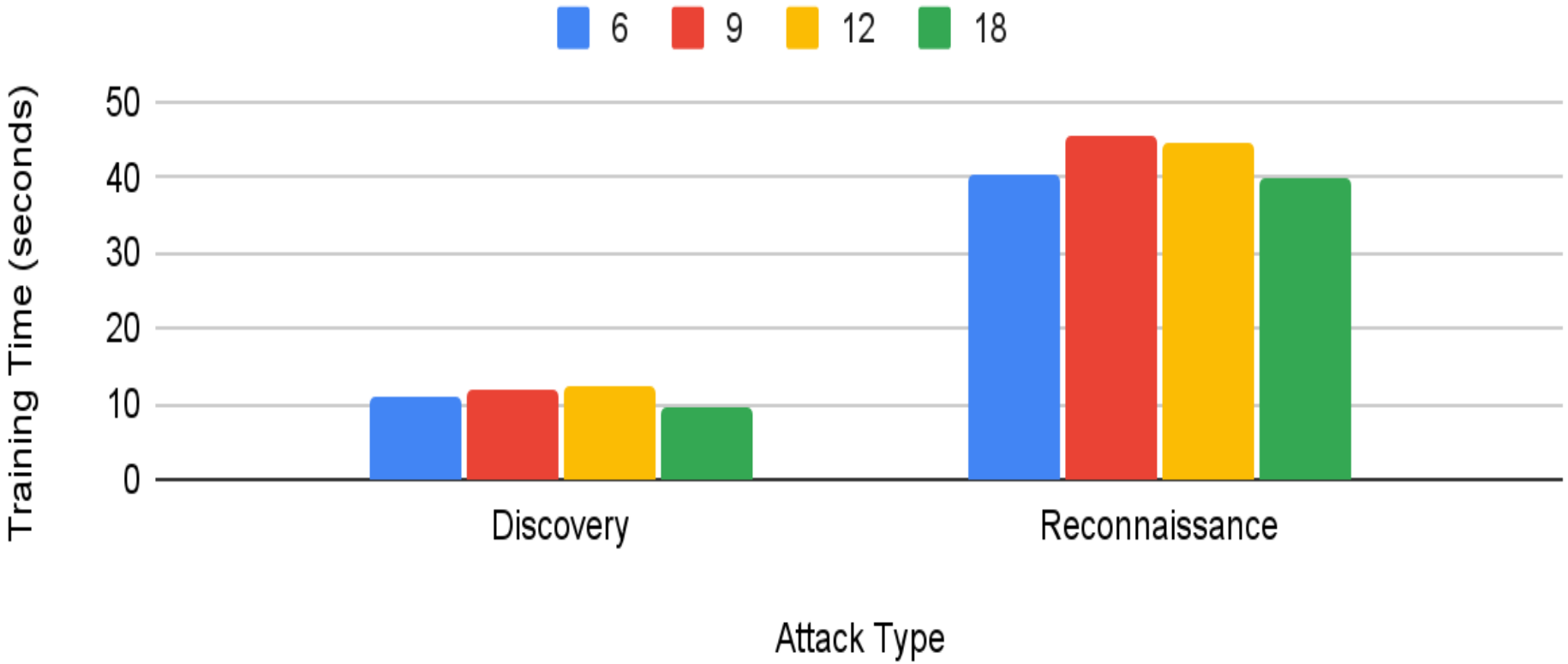
Config Setting	Value
Driver Cores	2
Driver Memory	10g
Executor Instances	6
Executor Cores	6
Executor Memory	6
Shuffle Partitions	6

Spark's Resources and Total Allocations

Resources	Total Allocation
Driver Cores	2
Driver Memory	10g
Executor Total Cores	96
Executor Total Memory	120g

UNIVERSITY of WEST FLORIDA

Number of Attributes vs Training Times for all Algorithms



UNIVERSITY of WEST FLORIDA

Machine Learning Algorithms

- Binary Classification
  - Decision Tree
  - Gradient Boosting Tree
  - Random Forest
  - Naïve Bayes
  - Logistic Regression
  - SVM

UNIVERSITY of WEST FLORIDA

Experimentation >

6, 9, 12 and 18 features are compared

For the two Tactics, Reconnaissance and Discovery

Using Spark's optimum Parameters

UNIVERSITY of WEST FLORIDA

Spark's Parameters Used

Machine Learning Algorithm	Parameters used
Logistic Regression	featuresCol: str = 'features', labelCol: str = 'label', predictionCol: str = 'prediction', maxIter: int = 10, regParam: float = 0.3, elasticNetParam: float = 0.8, tol: float = 1e-06, fitIntercept: bool = True, threshold: float = 0.5, thresholds: Optional[List[float]] = None, probabilityCol: str = 'probability', rawPredictionCol: str = 'rawPrediction', standardization: bool = True, weightCol: Optional[str] = None, aggregationDepth: int = 2, family: str = 'auto', lowerBoundsOnCoefficients: Optional[pyspark.ml.linalg.Matrix] = None, upperBoundsOnCoefficients: Optional[pyspark.ml.linalg.Matrix] = None, lowerBoundsOnIntercepts: Optional[pyspark.ml.linalg.Vector] = None, upperBoundsOnIntercepts: Optional[pyspark.ml.linalg.Vector] = None, maxBlockSizeInMB: float = 0.0
Naive Bayes	featuresCol: str = 'features', labelCol: str = 'label', predictionCol: str = 'prediction', probabilityCol: str = 'probability', rawPredictionCol: str = 'rawPrediction', smoothing: float = 1.0, modelType: str = 'multinomial', thresholds: Optional[List[float]] = None, weightCol: Optional[str] = None
Random Forest	featuresCol: str = 'features', labelCol: str = 'label', predictionCol: str = 'prediction', probabilityCol: str = 'probability', rawPredictionCol: str = 'rawPrediction', maxDepth: int = 5, maxBins: int = 32, minInstancesPerNode: int = 1, minInfoGain: float = 0.0, maxMemoryInMB: int = 256, cacheNodeIds: bool = False, checkpointInterval: int = 10, impurity: str = 'gini', numTrees: int = 20, featureSubsetStrategy: str = 'auto', seed: Optional[int] = None, subsamplingRate: float = 1.0, leafCol: str = '', minWeightFractionPerNode: float = 0.0, weightCol: Optional[str] = None, bootstrap: Optional[bool] = True

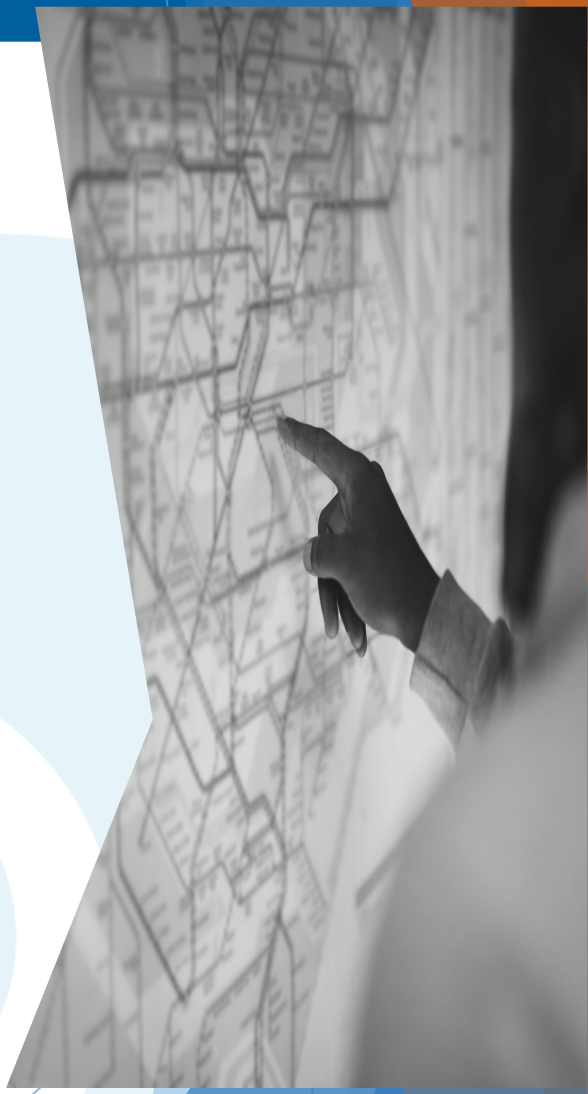
UNIVERSITY of WEST FLORIDA

Spark's Parameters Used

Machine Learning Algorithm	Parameters used
Gradient Boosted Decision Tree	featuresCol: str = 'features', labelCol: str = 'label', predictionCol: str = 'prediction', maxDepth: int = 5, maxBins: int = 32, minInstancesPerNode: int = 1, minInfoGain: float = 0.0, maxMemoryInMB: int = 256, cacheNodeIds: bool = False, checkpointInterval: int = 10, lossType: str = 'logistic', maxIter: int = 20, stepSize: float = 0.1, seed: Optional[int] = None, subsamplingRate: float = 1.0, impurity: str = 'variance', featureSubsetStrategy: str = 'all', validationTol: float = 0.01, validationIndicatorCol: Optional[str] = None, leafCol: str = '', minWeightFractionPerNode: float = 0.0, weightCol: Optional[str] = None
Decision Tree	featuresCol: str = 'features', labelCol: str = "label_bin", predictionCol: str = 'prediction', probabilityCol: str = 'probability', rawPredictionCol: str = 'rawPrediction', maxDepth: int = 30, maxBins: int = 100, minInstancesPerNode: int = 1, minInfoGain: float = 0.0, maxMemoryInMB: int = 256, cacheNodeIds: bool = False, checkpointInterval: int = 10, impurity: str = 'gini', seed: Optional[int] = None, weightCol: Optional[str] = None, leafCol: str = '', minWeightFractionPerNode: float = 0.0
SVM	featuresCol: str = 'features', labelCol: str = 'label_bin', predictionCol: str = 'prediction', maxIter: int = 100, regParam: float = 0.0, tol: float = 1e-06, rawPredictionCol: str = 'rawPrediction', fitIntercept: bool = True, standardization: bool = True, threshold: float = 0.0, weightCol: Optional[str] = None, aggregationDepth: int = 2, maxBlockSizeInMB: float = 0.0

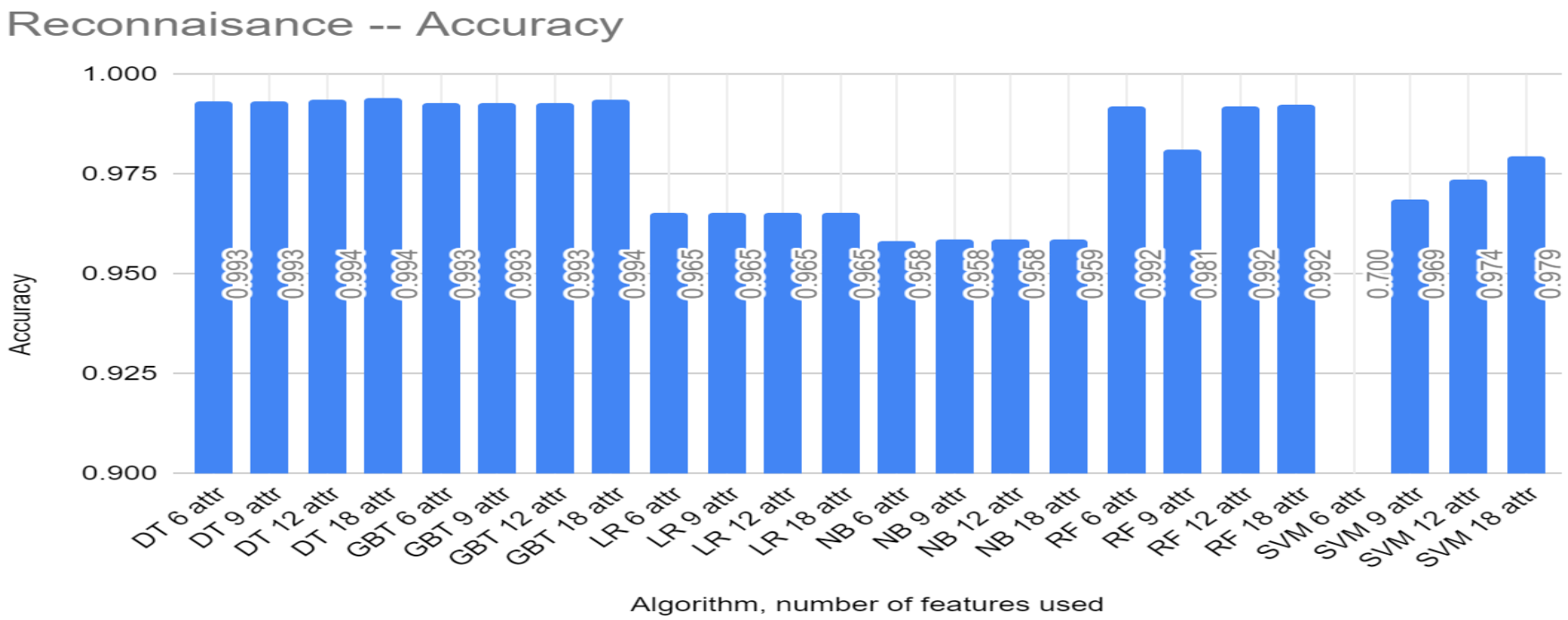
UNIVERSITY of WEST FLORIDA

The Reconnaissance Tactic



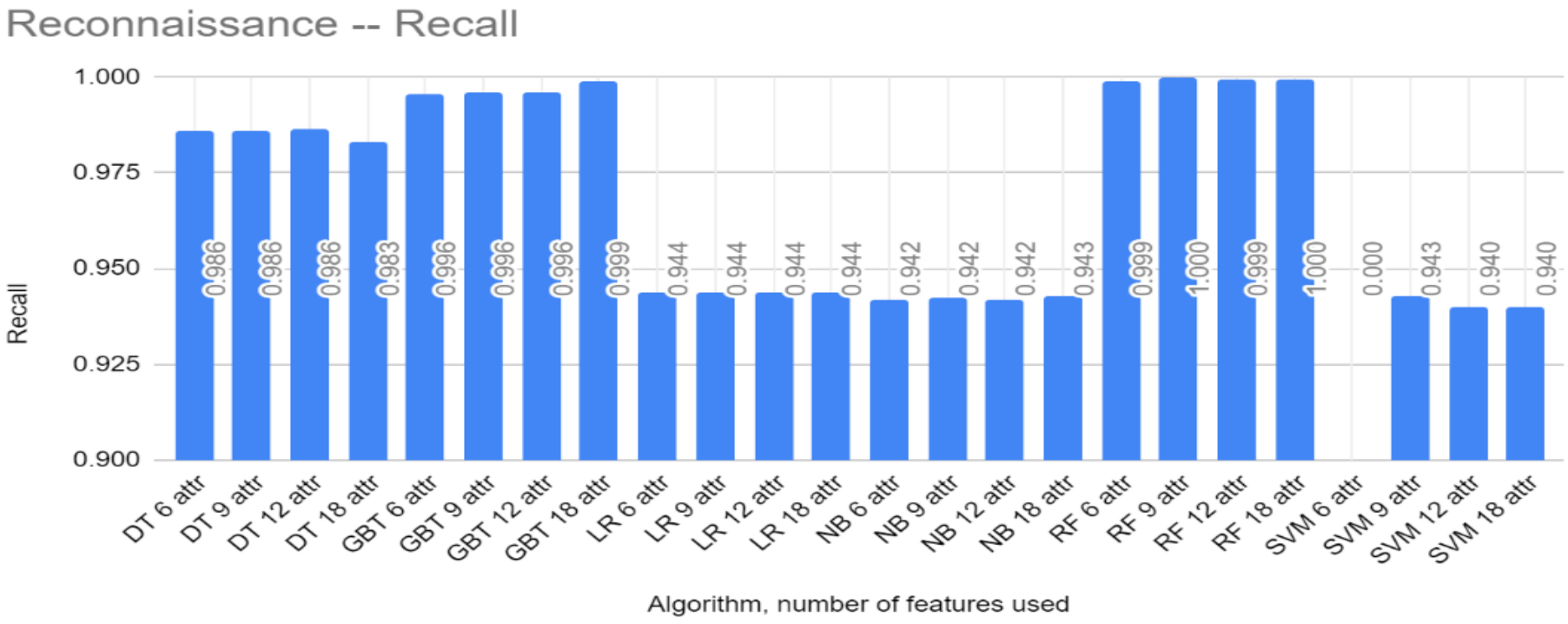
Reconnaissance: Accuracy - by Algorithms by Number of Features

- DT, GBT and RF had the highest averages for all sets of attributes. NB had the lowest accuracy, and LR was only slightly higher than NB.



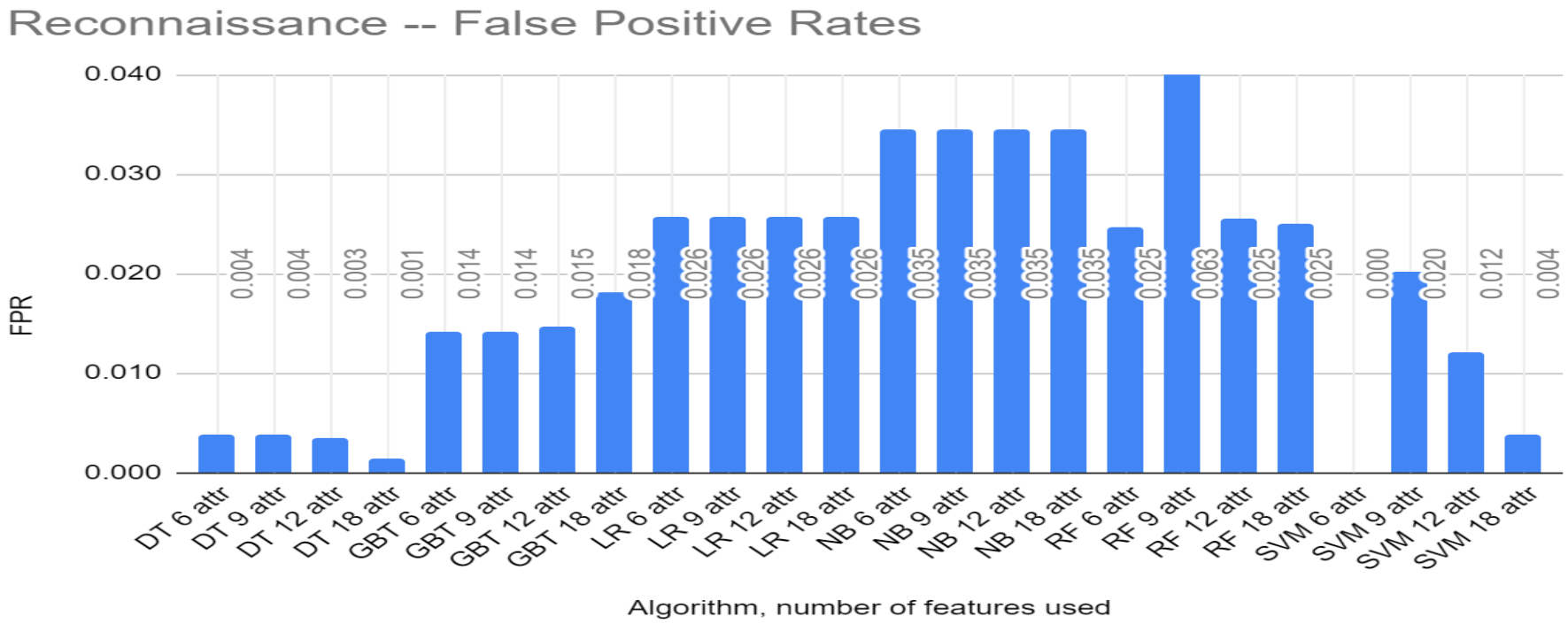
Reconnaissance: Recall by Algorithms by Number of Features

- GBT and RF had the highest recall, followed by DT. NB, SVM and LR.



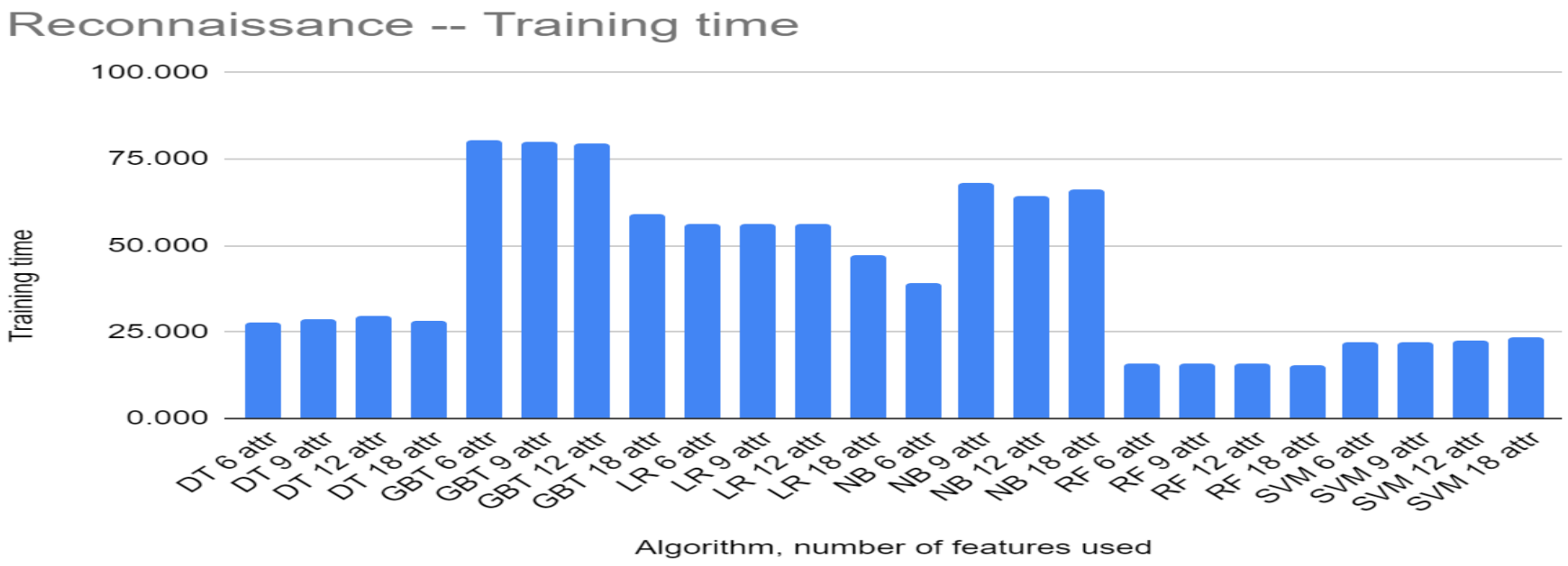
Reconnaissance: FPR by Algorithms by Number of Features

- DT had the lowest FPRs, and SVM and GBT seemed to have the second lowest FPRs. NB had the highest FPR rates



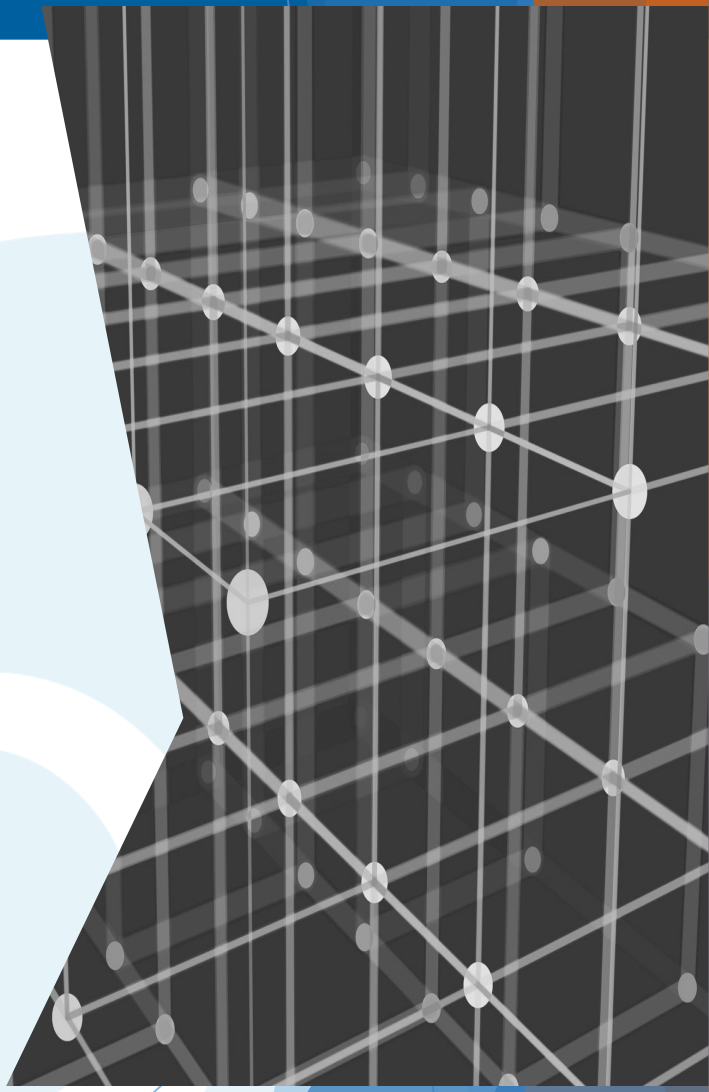
Reconnaissance: Training Time by Algorithms by Number of Features Used

- RF had the lowest training time, followed by DT, for all attribute combinations. GBT had the highest training times.



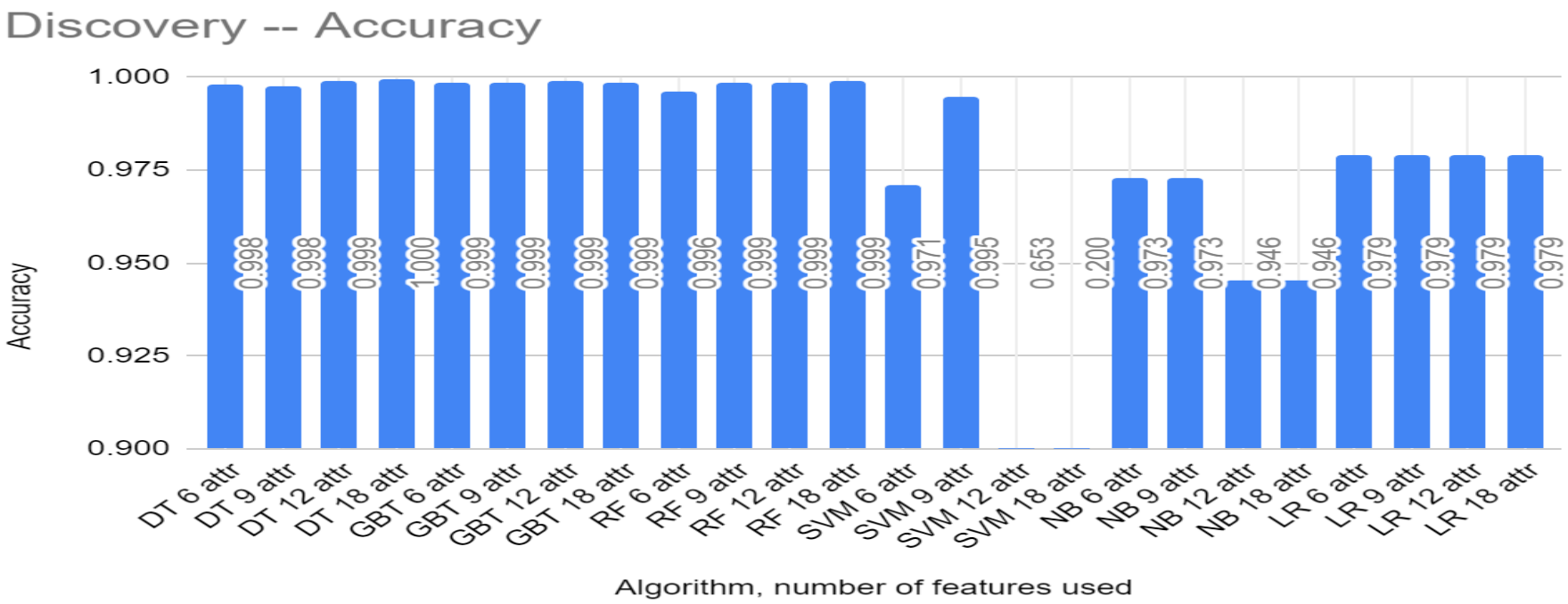
UNIVERSITY of WEST FLORIDA

The Discovery Tactic



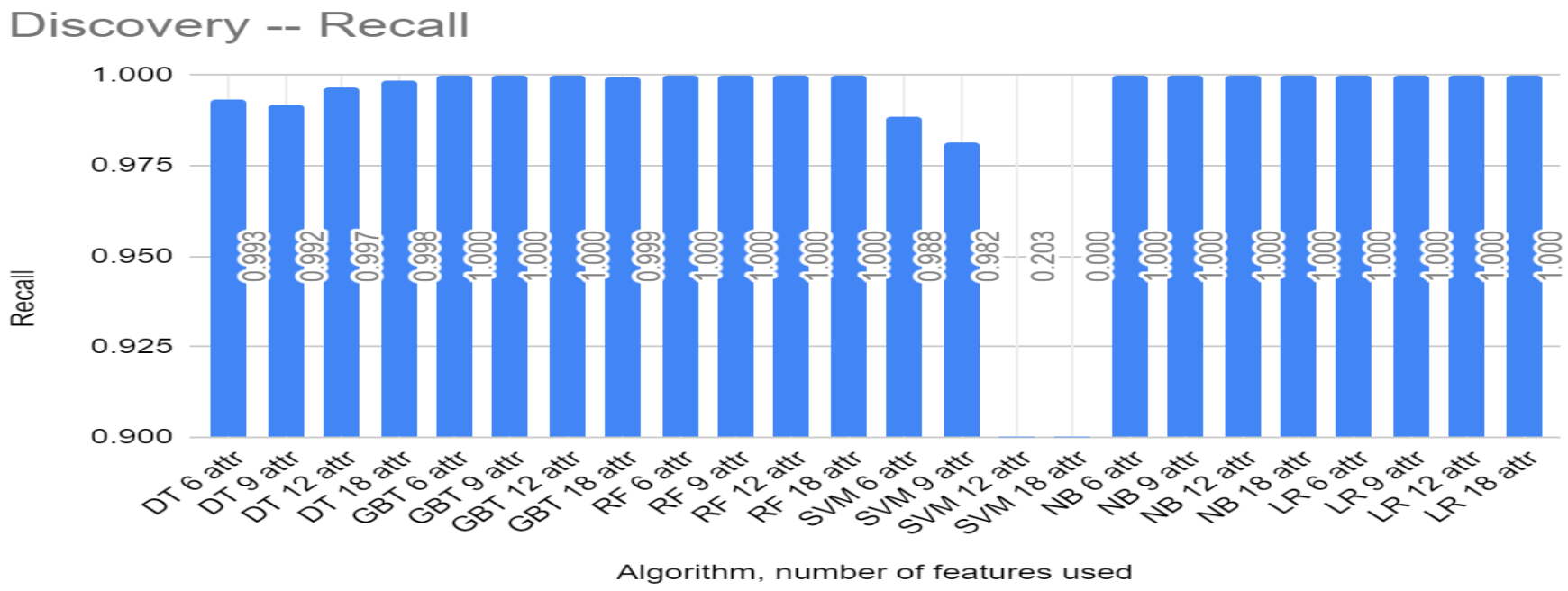
Discovery: Accuracy by Algorithms by Number of Features

- DT, GBT and RF had higher accuracy for all sets of attributes than the other three algorithms, SVM, NB and LR.



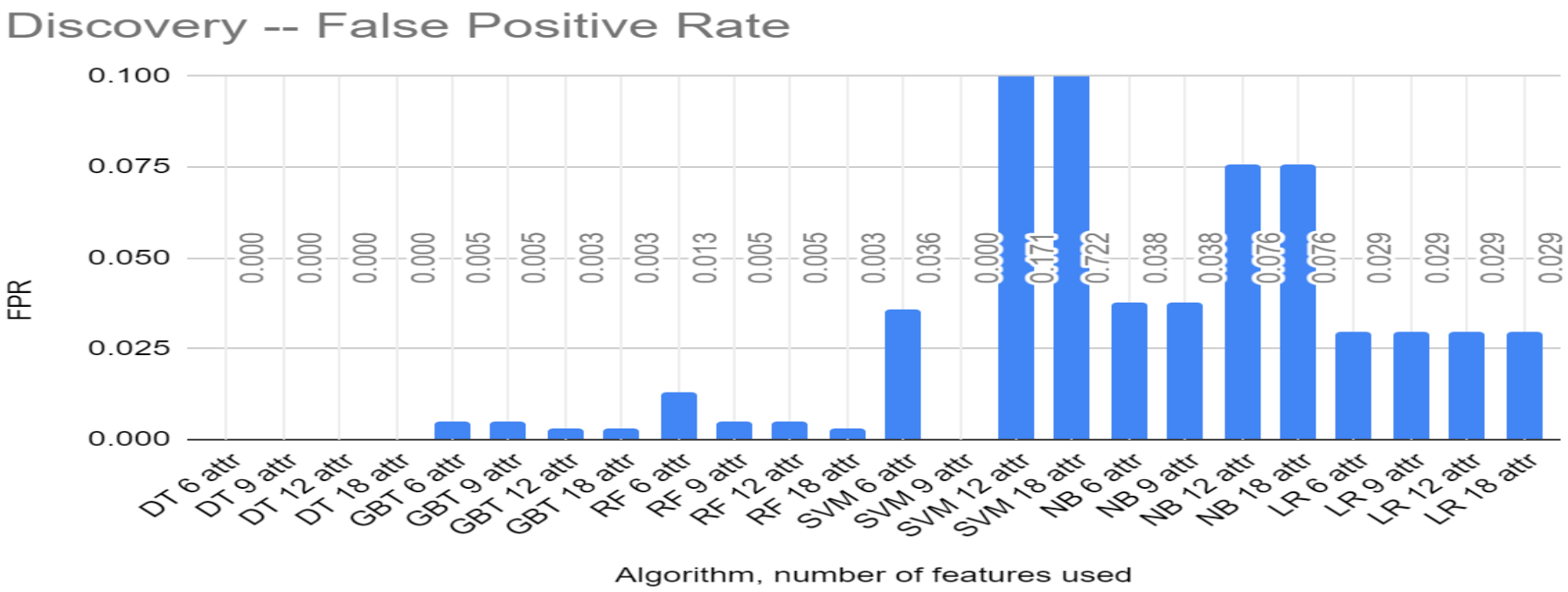
Discovery: Recall by Algorithms by Number of Features

- GBT, NB and LR had higher recall for all sets of attributes, and DT was close behind. SVM performed poorly in terms of recall.



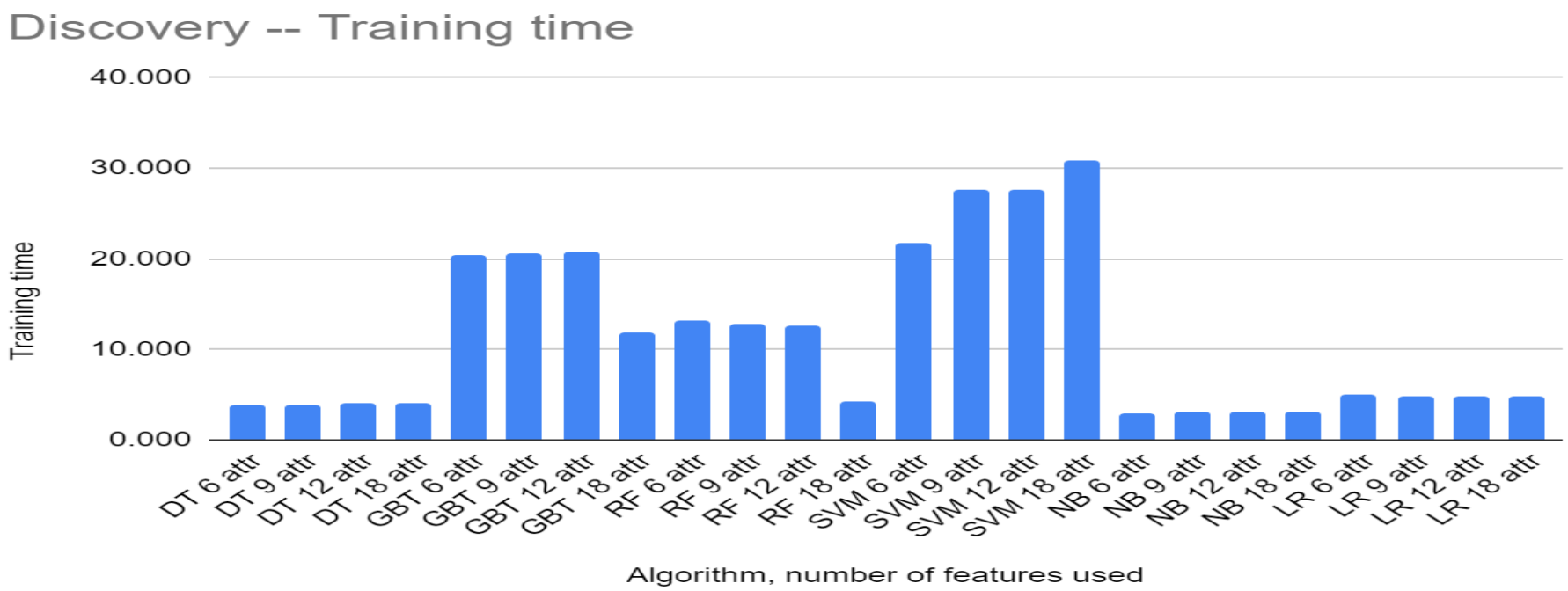
Discovery: FPR by Algorithms by Number of Features

- DT, GBT and RF all had lower FPRs for all combination of attributes, though DT appeared to perform the best.



Discovery: Training Time by Algorithms by Number of Features

- DT, NB, and LR had low training times. SVM and GBT had higher training times



UNIVERSITY of WEST FLORIDA

Conclusions

UNIVERSITY of WEST FLORIDA

Optimizing Classifier Performance on Spark

- More total cores for spark application makes ML algorithms run faster, but there are diminishing returns after a certain point
- Classifiers run fastest when the number of shuffle partitions is the same as the total number of executors
- There was no significant correlation between runtimes and the total amount of memory allocated (though allocating too little memory can cause executors to crash).

Machine Learning Results

- Tree-based methods (DT, GBT, RF) performed better on most metrics than the other three algorithms in classifying this dataset, for both the Renaissance and Discovery tactics.
  - These three algorithms all showed 99%+ accuracy for both attack tactics, with similarly higher scores in precision, recall, f-measure, and AUROC.
  - GBT and RF performed a little better than DT in terms of recall for both the tactics but in terms of the FPR
  - DT had the lowest FPRs for both Reconnaissance and Discovery

Machine Learning Results Cont'.

- Training times -- RF performed the best for Reconnaissance, followed by DT; for Discovery, DT performed the best.
- Best number of features -- the top 6 features from information gain -- history, protocol, service, orig_bytes, dest_ip, orig_pkts

References

- Huong, T. T., Bac, T. P., Long, D. M., Thang, B. D., Binh, N. T., Luong, T. D., & Phuc, T. K. (2021). LocKedge: Low-Complexity Cyberattack Detection in IoT Edge Computing. IEEE Access, 9, 29696–29710. https://doi.org/10.1109/access.2021.3058528
- *Zeek: About*. (2020). The Zeek Project. https://zeek.org/about/ (accessed February 2022)
- University of West Florida (2022) https://datasets.uwf.edu/
- What Is the MITRE ATT&CK Framework? | Get the 101 Guide. (2022). Trellix. https://www.trellix.com/en-us/security-awareness/cybersecurity/what-is-mitre-attack-framework.html (accessed February 2022)
- MITRE ATT&CK (2022) Reconnaissance, Tactic TA0043 - Enterprise | MITRE ATT&CK®. https://attack.mitre.org/tactics/TA0043/ (accessed February 2022)
- MITRE ATT&CK (2022) Discovery, Tactic TA0007 - Enterprise | MITRE ATT&CK®. https://attack.mitre.org/tactics/TA0007/ (accessed February 2022)
- The Zeek Project. (2022). *base/protocols/conn/main.zeek — Book of Zeek (v5.0.0)*. Zeek. https://docs.zeek.org/en/v5.0.0/scripts/base/protocols/conn/main.zeek.html (accessed February 2022)
- Kala Karun, A., & Chitharanjan, K. (2013). A review on hadoop; HDFS infrastructure extensions. 2013 IEEE Conference on Information and Communication Technologies. https://doi.org/10.1109/cict.2013.6558077
- Belouch, M., el Hadaj, S., & Idhammad, M. (2018). Performance evaluation of intrusion detection based on machine learning using Apache Spark. Procedia Computer Science, 127, 1–6. https://doi.org/10.1016/j.procs.2018.01.091
- Gupta, G. P., & Kulariya, M. (2016). A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark. Procedia Computer Science, 93, 824–831. https://doi.org/10.1016/j.procs.2016.07.238

References

- Morfino, V., & Rampone, S. (2020). Towards Near-Real-Time Intrusion Detection for IoT Devices using Supervised Learning and Apache Spark. Electronics, 9(3), 444. https://doi.org/10.3390/electronics9030444
- Malik, A.J., Khan, F.A. A (2018) hybrid technique using binary particle swarm optimization and decision tree pruning for network intrusion detection. Cluster Computing 21, 667–680. https://doi.org/10.1007/s10586-017-0971-8
- Kevric, J., Jukic, S. & Subasi, A. (2017) An effective combining classifier approach using tree algorithms for network intrusion detection. Neural Computing & Applications 28, 1051–1058. https://doi.org/10.1007/s00521-016-2418-1
- Zhang, J., Sun, J. & He, H. (2021) Clustering Detection Method of Network Intrusion Feature Based on Support Vector Machine and LCA Block Algorithm. Wireless Personal Communications. https://doi.org/10.1007/s11277-021-08353-y
- Du, R., Li, Y., Liang, X. et al. (2022) Support Vector Machine Intrusion Detection Scheme Based on Cloud-Fog Collaboration. Mobile Networks and Applications. https://doi.org/10.1007/s11036-021-01838-x
- Leevy, J. L., Hancock, J., Zuech, R., & Khoshgoftaar, T. M. (2021). Detecting cybersecurity attacks across different network features and learners. Journal of Big Data, 8(1). https://doi.org/10.1186/s40537-021-00426-w
- Sharafaldin, I., Habibi Lashkari, A., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. Proceedings of the 4th International Conference on Information Systems Security and Privacy. https://doi.org/10.5220/0006639801080116
- Microsoft. (2012, July 18). *Address Classes*. Microsoft Docs. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc940018(v=technet.10)?redirectedfrom=MSDN (accessed April 2022)
- Bagui, S., Simonds, J., Plenkers, R., Bennett, T. A., & Bagui, S. (2021). Classifying UNSW-NB15 network traffic in the Big Data Framework using random forest in Spark. *International Journal of Big Data Intelligence and Applications*, 2(1), 39–61. https://doi.org/10.4018/ijbdia.287617
- Rostami, S., Kleszcz, A., Dimanov, D., & Katos, V. (2020). A machine learning approach to dataset imputation for software vulnerabilities. *Communications in Computer and Information Science*, 25–36. https://doi.org/10.1007/978-3-030-59000-0_3

Acknowledgements

- The research was supported by 2021 NCAE-C-002: Cyber Research Innovation Grant Program, Grant Number: H98230-21-1-0170

UNIVERSITY of WEST FLORIDA

Thank you!

UNIVERSITY of WEST FLORIDA

Questions ?

Data Available at: datasets.uwf.edu