Part 3. Machine Learning

1. Machine learning algorithms

This report has been commissioned as part of an ongoing initiative to improve the security posture of ScottishGlen, an emerging company in the energy sector. The focus will be on exploring how machine learning (ML) could be leveraged to improve network security within the organisation, specifically with a view to performing automated network traffic analysis.

ML techniques are seeing rapid expansion in cybersecurity practices, offering improved performance and accuracy over the more traditional human-led approach in detecting malicious activities (Shaukat et al., 2020). These techniques can offer significant benefits in analysing complex, multi-featured network traffic flow data when compared to individual analysts, demonstrating higher levels of success in identifying subtle and nuanced patterns within the data (Shen et al., 2022). When designing an ML implementation, it can be easy to become overwhelmed by the range of algorithms available, making it difficult to ensure an appropriate model is used (Neo, 2021). Figure 1 presents a high-level diagram of the decision process for choosing the type of model:

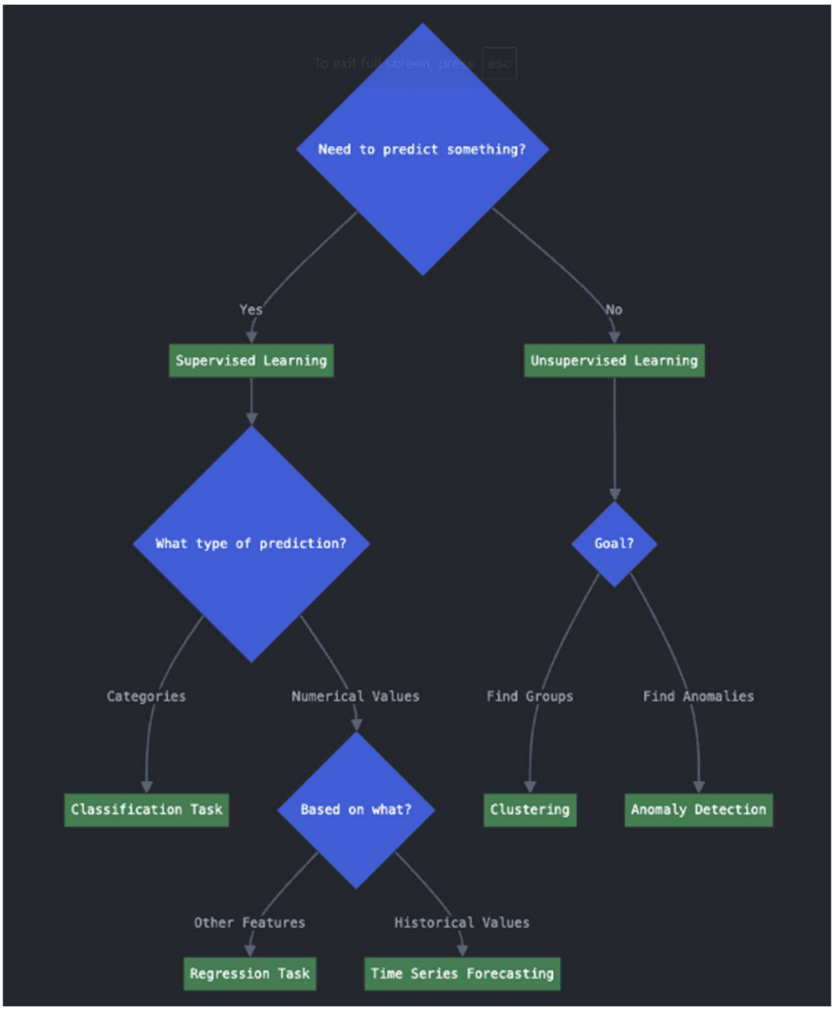

Figure 1: decision flowchart for identifying an ML model (Carrascosa, 2024).

As demonstrated in the figure above, ML models can be broadly categorised based on the context of the problem they are required to solve and the types of data they use to learn from. A detailed overview of these categories, including some common algorithms used per category, can be found in Appendix A.

2. Machine learning model design phases

The process of designing and building an effective ML model can be divided into seven stages, illustrated in Figure 2. This is sometimes referred to as the Machine Learning Development Life Cycle (MLDLC).

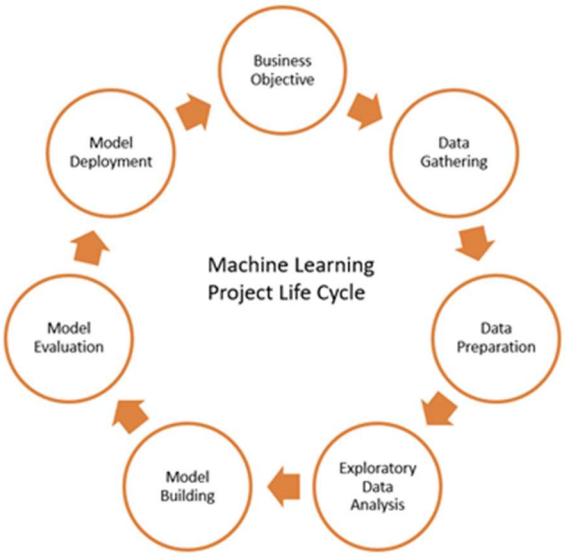

Figure 2: the stages of the Machine Learning Development Life Cycle (Paul, 2020).

A brief description of each stage is given below:

- Business objective – define the problem that the organisation wants to "solve" with the use of ML (e.g. "categorise incoming e-mails as spam or not spam").
- Data gathering – gather real-world data that will be used to train the model.
- Data preparation – prepare the data for use in the model. Crucially, this stage includes (among other tasks) handling missing values and addressing outliers in the dataset. It is arguably the most time-consuming stage of the process (Data-Driven Science, 2020).
- Exploratory data analysis (EDA) – identify patterns and trends in the data to inform decision making later in the process. Visualisation of the data (with charts and graphs) is typically utilised to assist with the analysis.
- Model building – this stage encompasses both the feature engineering/scaling and model training tasks. Features are selected, modified, and scaled (to ensure equality of their contribution) before being fed into a model for training. The dataset is split into training and testing sets in this stage.
- Model evaluation – test the success of the model using the test dataset and choose the metrics to be used for analysis, evaluating the model in line with the results. If the model is not performing as required, it is possible to return to an earlier stage of the process for tuning and correction.
- Model deployment – implement the model into the production environment once its performance is deemed satisfactory.

3. Model implementation

This report will now discuss the implementation of a machine learning model in line with the stages of the MLDLC outlined above. The model was built using Python, which is the most popular programming language to use for ML implementations owing to it being easy to use/read and the wealth of libraries available (Karl, 2023). A breakdown of the libraries used and justification for their use can be found in Appendix B.

3.1 Business objective

It was decided that an automated system that could identify suspicious traffic by examining network logs (flagging potentially malicious flows for further analysis by technical staff) would assist in improving ScottishGlen's cybersecurity posture.

3.2 Data gathering

Data was collected by technical staff members, consisting of historical network traffic logs containing metrics and some aggregated statistics.

3.3 Data preparation

There were no null values present in the data (illustrated in Figure 3), which was cleaned by the technical staff after collection. All observations in the data were labelled with the "correct" classification category.

```
print(df.isnull().sum())

Checking for missing values in the dataset:
dur                0
proto              0
service            0
state              0
spkts              0
dpkts              0
sbytes             0
dbytes             0
rate               0
sttl               0
dttl               0
sload              0
dload              0
sloss              0
dloss              0
sinpkt             0
dinpkt             0
sjit               0
djit               0
swin               0
stcpb              0
dtcpb              0
dwin               0
tcprtt             0
synack             0
ackdat             0
smean              0
dmean              0
trans_depth        0
response_body_len  0
ct_srv_src         0
ct_state_ttl       0
ct_dst_ltm         0
ct_src_dport_ltm   0
ct_dst_sport_ltm   0
ct_dst_src_ltm     0
is_ftp_login       0
ct_ftp_cmd         0
ct_flw_http_mthd   0
ct_src_ltm         0
ct_srv_dst         0
is_sm_ips_ports    0
attack_cat         0
dtype: int64
```

Figure 3: terminal output displaying counts of null values in the dataset.

3.4 Exploratory data analysis (EDA)

The following information was ascertained during the EDA stage (using the training dataset for analysis):

- The “attack_cat” variable was identified as the target feature containing the labels. There were ten possible outcomes, eight of which were indicative of suspicious traffic.
- The training dataset consisted of 175340 observations with 42 variables.
- With the exception of the target variable and further three categorical variables (“proto”, “service”, and “state"), all other features held numeric values.
- An analysis of the values in the variables identified two features (“is_ftp_login” and “is_sm_ips_ports”) that were binary categorical variables labelled as “0” for false and “1” for true (see Appendix C).
- All continuous variables used different scales for the measurements, suggesting that the data may need to be scaled for use when training the model (see Appendix C).
- Some of the continuous variables were heavily skewed, suggesting they might be of limited use when training the model (see Appendix C).

The code used to perform the EDA can be found in the supporting file “`dataset_exploration.py`”.

3.5 Model building

The first step in building the model is to categorise its purpose as this informs the rest of the design and implementation process. Respecting the flow chart process in Figure 1 resulted in the decision process that follows:

1. *Need to predict something?* **Yes** – network traffic should be predicted as either benign or malicious based on known network traffic metrics. This provided confirmation that a **supervised** learning model should be used.
2. *What type of prediction?* **Categorical** – network traffic should be labelled with one of ten string values. This provided confirmation that the model was required to perform a **classification** task. There are more than two possible outcomes for the target label, so this task is further categorised as a **multi-class classification** task.

It is standard practice to split the entire dataset available into a dataset for training the model and one for testing the model prior to any feature scaling or engineering taking place. This step had already been completed by the technical staff. The testing set contained a further 82326 values, representing an approximate 70/30 split of the entire dataset across the training and testing sets respectively. This is a common ratio used in ML and allows for more rigorous evaluation of the model (Gunkurnia, 2024).

As discovered during the EDA phase, the dataset provided consisted of a large number of features (42), all with different scales of measurement. In an attempt to proactively address potential issues with the speed of the model, the training dataset went through some feature scaling and engineering processes. Feature scaling is a pre-processing technique that aims to bring data values closer to one another, enabling a model to find more meaningful correlations in a short period of time (Sharma, 2022) and allowing for those correlations to be evaluated more fairly (Baravaliya, 2023). The standardisation applied to the training set was a z-score standardisation, which transforms the values so that they have 0 as their mean and variance. This standardisation technique is suitable for values that have non-normal distributions (Baravaliya, 2023), as demonstrated in Appendix B. The mathematical formula for z-score standardisation is as follows:

$$z = (x - u) / s$$

where *z* is equal to the standardised value, *x* is the original non-standardised value, *u* is the mean, and *s* is the standard deviation (both respective to that particular variable's values). The formula can be applied to data in an automated manner using the StandardScaler function from Scikit-learn. An example of the difference between the pre-standardisation and post-standardisation values can be seen in Figure 4.

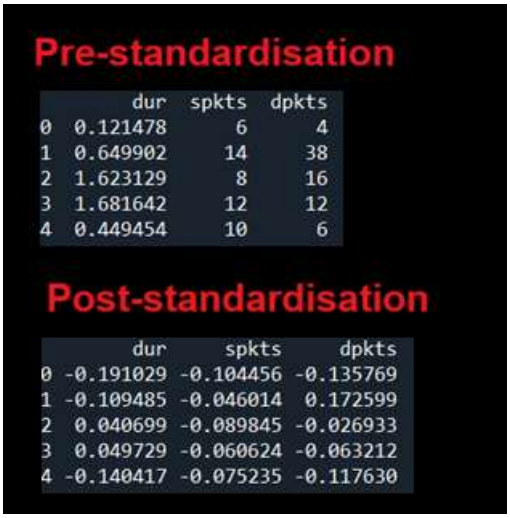

Figure 4: pre- and post-standardisation values.

Prior to the standardisation being performed, the training dataset was divided to remove the categorical values (including the labelling variable), as standardisation is not an applicable technique to apply to non-numerical values.

Returning to the findings of the EDA, it was evident that there were several variables that were heavily skewed, showing little variance and suggesting that they would be of limited use when training a model (Garg, 2021). Any variables with a variance of less than one (this threshold chosen because the data had already been z-score standardised) were removed used the VarianceThreshold function from Scikit-learn. Any remaining variables were then joined back together to the categorical variables previously segregated to construct the standardised and engineered dataset. Removing any features with a low variance resulted in a reduction in features to be used in training the model from 41 to 24. The final dataset was then extracted to a .csv file that could be used in model training. The code used to perform the engineering and scaling can be found in the supporting file "`feature_eng_scaling.py`". The resulting standardised data can be found in the supporting file "`standardised_data.csv`".

Random forest classifiers are commonly used for ML tasks where each observation is labelled once with one of a set of more than two possible outcomes (Jain, Ashish, and Mitra, 2022), as is the case for ScottishGlen. They typically have a high level of accuracy with large datasets whilst providing flexibility for tuning and the ability to understand feature importance easily (IBM, 2025). Random forests are not capable of using non-numerical values in the datasets that they use for testing and training (with the exception of the target variable), so these must be encoded prior to model implementation. In the case of the data that were provided by the technical staff, there were three columns that required this treatment: "`proto`", "`service`", and "`state`". Once these columns had been transformed, it became apparent that this had resulted in an inconsistency between the numbers of columns between the training and testing datasets, which is incompatible with the model

design. Given the high number of features already existing in the dataset, the additional columns found in the training dataset were identified and removed. The final datasets were then extracted to .csv files that could be used in model implementation. The code used to achieve this encoding can be found in the supporting file "`cat_encoding.py`". The resulting encoded data can be found in the supporting files "`encoded_train_data.csv`" and "`encoded_test_data.csv`".

Once the data pre-processing was complete, it was possible to commence work on training the model. The data was split into estimator and target variable datasets and a random forest classifier fit to the training data using the RandomForestClassifer function of Scikit-learn. The fitted model was then applied to the test dataset to evaluate its performance. The code used to achieve this can be found in the supporting file "`rf_standard.py`".

3.6 Model evaluation

There are a number of common metrics that can be used to evaluate the performance of an ML classifier:

- Confusion matrix.
- Accuracy.
- Recall.
- Precision.
- F1 score.
- Receiver operating characteristic (ROC curve).
- Precision recall curve).

A detailed overview of these categories can be found in Appendix D. Once calculated, these metrics can assist in the decision-making process around whether the model's performance could be improved by tuning its hyperparameters (variables containing configuration settings for a model). Given the number of possible outputs for this model (ten), it was deemed that other metrics would be more effective at evaluating this model than a confusion matrix, so this was not performed. The accuracy, recall, precision, and F1 scores were calculated using the ClassifierReport function of Scikit-learn, the results of which can be seen in Figure 5:

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	677
Backdoor	0.00	0.00	0.00	583
DoS	0.29	0.00	0.00	4089
Exploits	0.38	0.71	0.49	11131
Fuzzers	0.17	0.18	0.17	6061
Generic	0.99	0.96	0.98	18871
Normal	0.76	0.76	0.76	36998
Reconnaissance	0.69	0.01	0.01	3495
Shellcode	0.00	0.00	0.00	378
Worms	0.00	0.00	0.00	44
accuracy			0.67	82327
macro avg	0.33	0.26	0.24	82327
weighted avg	0.68	0.67	0.65	82327

Figure 5: accuracy, precision, recall, and F1 scores produced by the model.

The receiver operating characteristic (ROC) and precision-recall curves, typically being used for binary output, both required some further data transformation before they could be plotted. it was

necessary for all output labels to be binarised prior to the curves being calculated. This was achieved using the label_binarize function of Scikit-learn. Furthermore, it was necessary for the random forest model to be subjected to further configuration to enable binarised outputs. This was achieved using the OneVsRestClassifier function from Scikit-learn. It was also apparent from the EDA that there was an imbalance in the categorisation labels in the training dataset, so micro-averaging was deemed appropriate for the calculations involved in plotting the curves. This technique combines the predictions across all classes to allow for a more balanced representation of imbalanced classes. The ROC curve and area under curve (AUC) were calculated using the roc_curve and auc functions of Scikit-learn respectively. Finally, the curve was plotted using matplotlib. It is presented here as Figure 6:

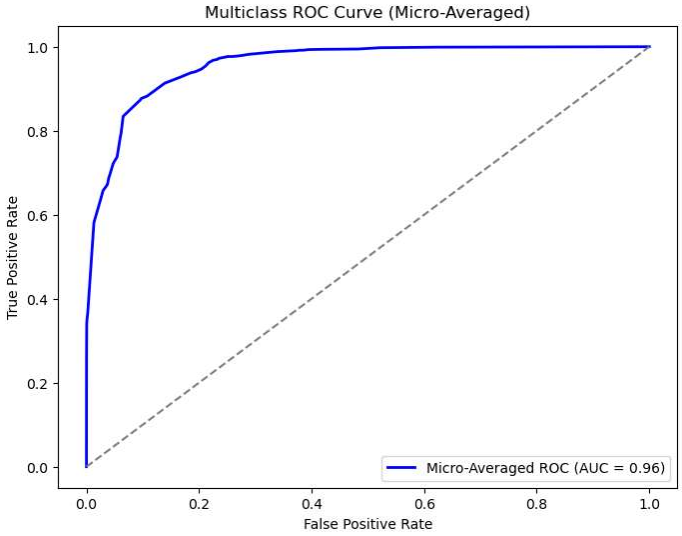

Figure 6: ROC curve and AUC score produced by the model.

The binarised output labels and micro-averaging techniques were similarly applied to the plotting of the precision-recall curve, which was achieved using the precision_recall_curve function of Scikit-learn. The average precision score was calculated using the average_precision_score Scikit-learn function and the curve plotted with matplotlib. The curve is presented here as Figure 7:

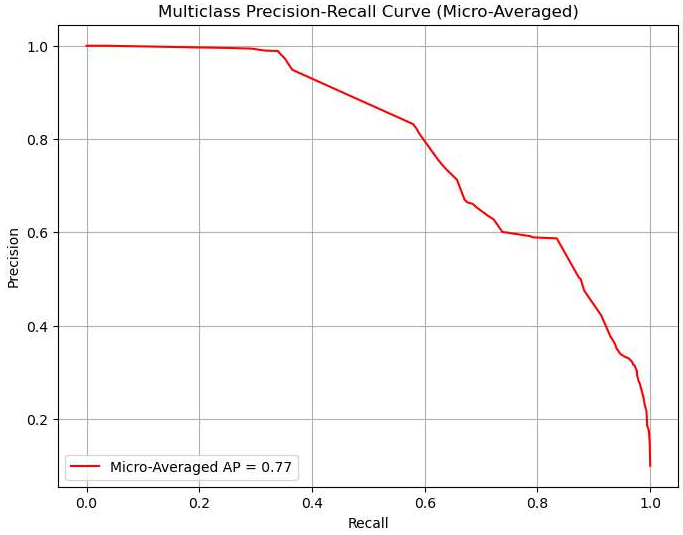

Figure 7: precision-recall curve and micro-averaged AP score produced by the model.

The industry standard for an acceptable accuracy score for ML models is generally held to be between 70-90% (Barkven, 2025) and varies between use cases (e.g. it is more important for a medical diagnosis tool to be highly accurate than a spam e-mail classifier). The weighted average scores for this model were found to fall short of this figure, however this is not necessarily a good measure of performance in models with imbalanced datasets (Evidently AI, 2025) as was the case with ScottishGlen. ROC or precision-recall curves can present a more accurate picture in these cases, with the latter providing a more reliable interpretation for imbalanced datasets. As such, the micro-averaged AP of 77% should be taken as a truer indicator than the 96% AUC score.

3.7 Model deployment

At the levels described in the model evaluation section, the model could be considered performant enough for its purpose of network traffic inspection and was deemed fit for deployment. The finished model, complete with the code used to carry out the evaluation of it, can be found in the accompanying file "`final_rf.py`".

References

Baravaliya, Y. (2023) *Feature Scaling and Normalization.* Available at: https://medium.com/@yashbaravaliya206/feature-scaling-and-normalization-ca484a16882a (Accessed: 19 April 2025).

Barkven, K. (2025) *How To Know if Your Machine Learning Model Has Good Performance.* Available at: https://www.zams.com/blog/machine-learning-model-performance (Accessed: 20 April 2025).

Carrscosa, I.P. (2024) *A Practical Guide to Choosing the Right Algorithm for Your Problem: From Regression to Neural Networks.* Available at: https://machinelearningmastery.com/practical-guide-choosing-right-algorithm-your-problem/ (Accessed: 15 April 2025).

Data-Driven Science (2020) *7 Stages of Machine Learning — A Framework.* Available at: https://medium.com/@datadrivenscience/7-stages-of-machine-learning-a-framework-33d39065e2c9 (Accessed: 16 April 2025).

Doshi, S. (2020) *Commonsense Validation and Reasoning using Natural Language Processing.* MSc thesis. Cork Institute of Technology. Available at: https://www.researchgate.net/profile/Shrenik-Doshi/publication/341703036_Commonsense_Validation_and_Reasoning_using_Natural_Language_Processing/links/5ecfa4f992851c9c5e63b659/Commonsense-Validation-and-Reasoning-using-Natural-Language-Processing.pdf (Accessed: 15 April 2025).

Evidently AI (2025) *Accuracy vs. precision vs. recall in machine learning: what's the difference?* Available at: https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall (Accessed: 20 April 2025).

Garg, s. (2021) *Dropping Constant Features using VarianceThreshold: Feature Selection -1.* Available at: https://medium.com/nerd-for-tech/removing-constant-variables-feature-selection-463e2d6a30d9 (Accessed: 19 April 2025).

Gunkurnia (2024) *Choosing the Optimal Data Split for Machine Learning: 80/20 vs 70/30?* Available at: https://medium.com/@gunkurnia/choosing-the-optimal-data-split-for-machine-learning-80-20-vs-70-30-0fd266710236 (Accessed: 19 April 2025).

IBM (2025) *What is Random Forest?* Available at: https://www.ibm.com/think/topics/random-forest (Accessed: 19 April 2025).

Jain, V., Phophalia, A. and Mitra, S.K. (2022) 'HML-RF: Hybrid Multi-Label Random Forest', *IEEE Access,* 10, pp. 22902-22914 Available at: https://doi.org/10.1109/access.2022.3154420.

Karl, T. (2023) *6 Reasons Why Is Python Used for Machine Learning.* Available at: https://www.newhorizons.com/resources/blog/why-is-python-used-for-machine-learning (Accessed: 16 April 2025).

Mahesh, B. (2020) 'Machine Learning Algorithms - A Review', *International Journal of Science and Research (IJSR),* 9(1) Available at: https://doi.org/10.21275/art20203995.

Neo, B. (2021) *Top 5 Machine Learning Algorithms Explained.* Available at: https://medium.com/bitgrit-data-science-publication/top-5-machine-learning-algorithms-explained-d15234b627f7 (Accessed: 14 April 2025).

Ngo, G., Beard, R. and Chandra, R. (2022) 'Evolutionary bagging for ensemble learning', *Neurocomputing,* 510 Available at: https://doi.org/10.1016/j.neucom.2022.08.055.

Paul, N. (2020) *Machine Learning Project Life Cycle.* Available at: https://pianalytix.com/machine-learning-project-life-cycle/ (Accessed: 16 April 2025).

Sharma, V. (2022) 'A Study on Data Scaling Methods for Machine Learning', *International Journal for Global Academic & Scientific Research,* 1(1) Available at: https://doi.org/10.55938/ijgasr.v1i1.4.

Shaukat, K., Luo, S., Varadharajan, V., Hameed, I.A. and Xu, M. (2020) 'A Survey on Machine Learning Techniques for Cyber Security in the Last Decade', *IEEE Access,* 8, pp. 222310–222354 Available at: https://doi.org/10.1109/access.2020.3041951.

Shen, M., Ye, K., Liu, X., Zhu, L., Kang, J., Yu, S., Li, Q. and Xu, K. (2022) 'Machine Learning-Powered Encrypted Network Traffic Analysis: A Comprehensive Survey', *IEEE Communications Surveys & Tutorials,* 25(1), pp. 791–824 Available at: https://doi.org/10.1109/comst.2022.3208196.

Shrikant, S. (2023) *Understanding the Accuracy Score Metric's Limitations in the Data Science Classification Problems.* Available at: https://www.linkedin.com/pulse/understanding-accuracy-score-metrics-limitations-data-akshay-w/ (Accessed: 19 April 2025).

Tiwari, A. (2022) 'Supervised learning: From theory to applications', *Artificial Intelligence and Machine Learning for EDGE Computing,* 2022, pp. 23–32 Available at: https://doi.org/10.1016/b978-0-12-824054-0.00026-5.

Wakefield, K. (2023) *A guide to the types of machine learning algorithms and their applications.* Available at: https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html (Accessed: 15 April 2025).

Appendices

Appendix A – Overview of Machine Learning Models

A.1 Supervised learning

In supervised learning models, machines "learn" by example. The purpose of the model is to predict a label for each record in a dataset based upon observable patterns between one or more "estimator" features of the same data record. An input dataset is divided into training and test sets, with the training set provided to the model containing pre-labelled data indicating the correct outcome for each individual case. The test set is then used to validate the findings of the model, allowing evaluation and enabling tuning for improved performance. A diagram of the workflow for supervised models can be seen in Figure 8.

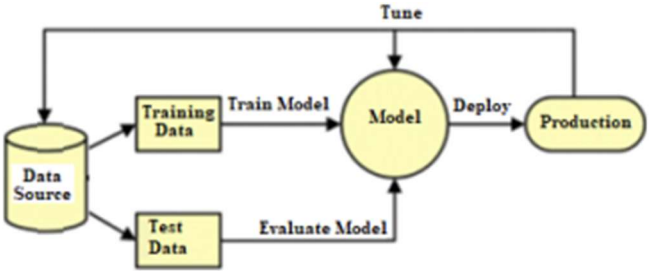

Figure 8: supervised learning workflow (Mahesh, 2020).

Supervised learning models are used for classification, regression, and forecasting tasks, and the choice of an appropriate algorithm relies heavily on which of these tasks the model will be required to perform (though other factors, such as computational efficiency and scalability, will also factor into the decision-making process).

A.1.1 Classification tasks

Classification tasks require existing data records to be labelled with a category. Classification tasks are very commonly used in cybersecurity, e.g. for categorising an email as "spam" or "not spam" (binary classification – two possible outcomes), or identifying a file as "ransomware", "trojan", or "safe" (multiclass classification). Classification tasks are very commonly used in cybersecurity. Some common algorithms used for classification tasks are (Wakefield, 2023.):

- Logistic regression – applies labels based on the estimated probability that a data record falls in one of two categories (binary classification) using the patterns found in previous data.
- Decision tree – applies a flow-chart-like technique to the data, testing each variable in a structured manner. Each node of the tree represents a specific test applied to the data, with outcomes represented by "branches". The resulting tree structure illustrates every possible outcome of each test. Decision trees can be implemented as a single tree or with multiple trees running concurrently using ensemble learning (known as random forests), which can improve the accuracy of the model.
- Support vector machines (SVM) – attempt to determine margins within the patterns in the data that can be used to separate records into the requisite categories.
- K-nearest neighbours – defines the differences and similarities between data records into "distances", allowing the model to categorise data that are within a stipulated distance of one another as belonging to the same class.

- Naïve Bayes – applies Bayes' theorem to predict the probability of a data record belonging to a class, whilst treating every feature within the dataset as being independent of others.

A.1.2 Regression tasks

In contrast to classification tasks, regression tasks aim to predict the numerical value of a dependant variable based on one or more "predictor" features in the dataset, e.g. predicting the appropriate rental value of a property based on one or more of the property's features. Regression tasks are not as common as those for classification in cybersecurity. The model types available for regression tasks are also fewer in number than those for classification tasks:

- Simple linear regression – predicts the value of a dependent variable based on its relationship with one other numeric feature (e.g. predicting the rental price of a property based on its square footage).
- Multiple linear regression - predicts the value of a dependent variable based on its relationship with more than one other feature of the same data record (e.g. predicting the rental price of a property based on its square footage, number of bedrooms, and building age).
- Decision trees (with or without random foresting) can also be used for regression tasks.

A.1.3 Forecasting tasks

The distinction between forecasting tasks and classification/regression tasks lies not in the type of predictions a model will make, but in the timeline of the predicted outcome. Forecasting models attempt to predict a category or value of a variable in the future, based on historic data (whereas classification/regression tasks look to apply labels to enrich data of past events). Forecasting can be used for classification (e.g. weather forecasting) or to predict numerical values, as with regression (e.g. predicting the cost of a data breach) and therefore can make use of the same models mentioned above.

A.2 Unsupervised learning for clustering/dimension reduction tasks

In unsupervised learning models, machines "learn" by exploring a dataset provided and identifying correlations and relationships within the data. The dataset provided to the model does not contain any pre-labelled data indicating the correct outcome for each individual case. The model then develops its findings when new data is presented to it, becoming more accurate and refined. The purpose of the model is to group or arrange data in a more organised way than when it was ingested. A diagram of the workflow for supervised models can be seen in Figure 9.

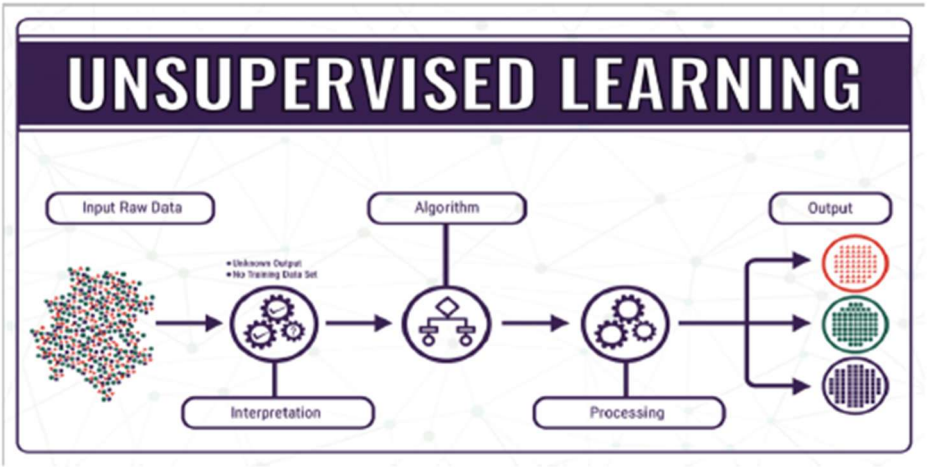

Figure 9: unsupervised learning workflow (Doshi, 2020).

Unsupervised learning models are typically used for clustering and feature (dimensionality) reduction tasks (Mahesh, 2020).

A.2.1 Clustering tasks

The purpose of clustering tasks is to group sets of similar data into clusters. Analysis is then typically performed on each cluster individually to find patterns, e.g. grouping URLs with similar structures for later identification as malicious or benign. A commonly used algorithm is K-means clustering, which identifies a "k centre" for each cluster using the data provided.

A.2.2 Dimensionality reduction tasks

Dimensionality reduction tasks aim to reduce the number of features being used in large, complex datasets by determining which sets of features may produce the greatest accuracy in an output, e.g. reducing the number of metrics to be considered for a network traffic analysis. Principal component analysis (PCA) is a commonly used algorithm used to perform dimensionality reduction, which can make recommendations on which features should be included in an analysis based on the correlations between sets of features in the dataset provided.

A.3 Ensemble methods

The models mentioned above are typically considered single stand-alone models. Ensemble methods combine multiple instances of these to achieve improved performance. The three most common forms of ensemble methods in ML are bagging, boosting, and stacking. Random forests are an enhancement of these ensemble methods (Ngo, Beard, and Chandra, 2022). They use multiple decision tree instances to converge on an outcome and can access any of these ensemble methods (among others) for improved accuracy and performance.

Appendix B - Libraries used in model

Name	Purpose
Pandas	Data analysis and manipulation. Can also return summary statistics and meta-information about imported data.
matplotlib.pyplot	Graph and chart generation for data visualisation.
Scitkit-learn functions	
StandardScaler	Standardisation of feature values.
VarianceThreshold	Identifies low variance features so they can be removed from the dataset prior to PCA.
RandomForestClassifier	Runs the random forest algorithm.
ClassificationReport	Calculates and returns accuracy, precision, recall, and F1 scores for a model.
label_binarize	Transforms multiclass output labels into binary classes.
OneVsRestClassifier	Splits a multiclass categorisation output into a binary output per class (for use in plotting the ROC and precision-recall curves).
roc_curve	Performs the calculations required for the plotting of the ROC curve.
auc	Calculates the AUC score of an ROC curve.
precision_recall_curve	Performs the calculations required for the plotting of the precision-recall curve.
average_precision_score	Calculates the average precision score for a model.

Appendix C – EDA histograms

Data presented in accompanying file “Appendix C – EDA Histograms” to allow for view zooming.

Appendix D – Evaluation metrics

D.1 Confusion matrix

A matrix that offers a simple view of an ML model's performance by counting the number of:

- True positives (TP).
- True negatives (TN).
- False positives (FP).
- False negatives (FN).

It is typically laid out as in Figure 10.

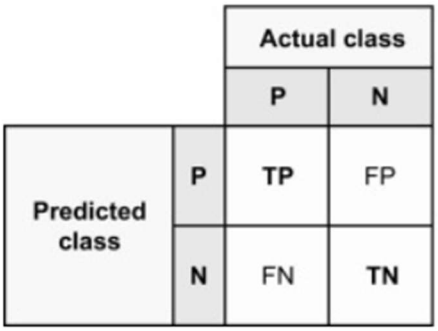

Figure 10: confusion matrix template (Tiwari, 2022).

They are commonly used for binary outputs but can be applied to multi-class classification tasks. In ML models where there are large numbers of possible output values, it may be more effective to use an alternative evaluation metric.

D.2 Accuracy score

The accuracy score is a calculation of the proportion of true positives out of all predictions. It is represented mathematically using the formula in Figure 11:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 11: mathematical formula for accuracy score (Shrikant, 2023).

D.3 Recall score

The recall score is a calculation of the proportion of true positives out of all actual positives. It is represented mathematically using the formula in Figure 12:

$$Recall = \frac{TP}{TP + FN}$$

Figure 12: mathematical formula for recall score (Shrikant, 2023).

D.4 Precision score

The precision score is a calculation of the proportion of true positives out of all positive predictions. It is represented mathematically using the formula in Figure 13:

$$Precision = \frac{TP}{TP + FP}$$

Figure 13: mathematical formula for precision score (Shrikant, 2023).

D.5 F1 score

The F1 score is a weighted average of the precision and recall scores combined. It is weighted to allow the F1 score to have a low value in instances where the precision or recall scores are 0. It is represented mathematically using the formula in Figure 14:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Figure 14: mathematical formula for F1 score (Shrikant, 2023).

D.6 Receiver operating characteristic (ROC) curve

The ROC curve (also known as the "area under the receiver operating characteristic (AUC-ROC) curve) is a measure of the model's ability to distinguish between classes. It is a plot of the true positive rate against the false positive rate at each threshold setting. The area under the curve is calculated to determine how well a model can distinguish between classes. The more capable a model, the closer the AUC value will be to 1. These curves are typically considered suitable in cases where there is little imbalance amongst the classes in a model's predictions.

D.7 Precision-recall curve

This curve shows the trade-off between the precision and recall at different thresholds, as it is common to see precision decrease as recall increases. A high value under the curve indicates a model has both high recall and high precision. These curves are typically considered suitable in cases where there is imbalance amongst the classes in a model's predictions. In cases where a precision-recall is used for multiclass classification the output must be binarised, or one curve should be plotted per label.