EVALUATING AND TESTING AI

SYLLABUS
Evaluating algorithms-Dealing with raw Data, Data Normalization, managing categorical variables. Detectors performance- ROC curve and AUC measure, split data in to training and test sets, testing for data and model quality, Ensuring security and reliability

EVALUATING ALGORITHMS

- Several AI solutions are available to achieve certain cybersecurity goals, so it is important to learn how to evaluate the effectiveness of various alternative solutions, using appropriate analysis metrics.
- At the same time, it is important to prevent phenomena such as overfitting, which can compromise the reliability of forecasts when switching from training data to test data. Evaluating algorithms in cybersecurity involves rigorous steps from preparing the data (feature engineering, splitting) and training the model (managing overfitting, cross-validation, hyperparameter tuning) to assessing its performance using appropriate metrics (ROC, AUC, Confusion Matrix).
- Furthermore, it requires considering the security landscape, including how attackers might try to evade detectors, and ensuring the overall quality, reliability, and security of the AI solution itself, particularly regarding data integrity, performance, and scalability.

DEALING WITH RAW DATA

- **Raw Data Nature**: Raw data consists of fragments of information, like pieces of a puzzle, and it's noted that data alone is rarely able to "speak for itself" and can be misleading.
- **Necessity of Transformation**: Raw data often needs to be transformed into numerical form to be usable by AI algorithms, which are frequently mathematical models.
- **Feature Engineering**: The process of converting raw data into numerical **features** and defining the features required by algorithms is called feature engineering. This is described as an essential preliminary process for achieving analytical goals and implementing efficient predictive models, as the quality and quantity of features impact predictive performance.
- **Initial Screening**: Dealing with raw data involves a first screening based on the required nature of numerical values for the models, such as whether they

should be positive, negative, Boolean, within certain magnitude ranges, or have predefined maximum/minimum values.

- **Data Binarization**:
  - This is described as one of the most basic forms of transformation, often applied to raw data counts.
  - It involves assigning the value **1** to all counts greater than 0 and **0** otherwise.
  - Its usefulness is illustrated with predicting user preferences based on video views: the raw count might not be reliable due to varying user habits (some repeatedly watch the same video, others focus and watch fewer times). Different users having orders of magnitude differences in views (tens to thousands) also makes statistical measures like the average less representative.
  - Instead of using the raw count, binarizing the counts (1 if viewed > 0, 0 otherwise) provides a more efficient and robust measure of individual preferences.
- **Data Binning**:
  - This technique addresses the problem of algorithms performing poorly when faced with data exhibiting a wide range of values, such as clustering algorithms using Euclidean distance for similarity measurement.
  - Similar to binarization, it aims to reduce the dimensional scale.
  - It involves grouping raw data counts into **bins** (containers), typically with fixed amplitude and sorted in ascending order. This scales down their absolute values, either linearly or exponentially.
- **Logarithmic Data Transformation**:
  - This transformation involves replacing the absolute values of raw data counts with their logarithms.
  - It has the peculiar feature of reducing the relevance of greater values while simultaneously amplifying smaller ones.
  - This results in a greater uniformity of value distribution.
  - Other power functions, such as the Box–Cox transformation, can also be used to stabilize the variance of a data distribution.
- **Importance of Data Quality**: Ensuring data quality is critical. Preliminary selection and cleaning are necessary to eliminate irrelevant or redundant information and present the data to algorithms in a correct form, which can improve their learning ability. The principle of **GIGO (garbage in, garbage out)** is mentioned, implying that the reliability of predictions depends on data quality. This includes making sure the data is reliable and complete before feeding it to algorithms.

DATA NORMALIZATION

Data normalization, also referred to as feature normalization or feature scaling, is a preprocessing technique used to standardize the range of independent variables or features of data. It plays a crucial role in improving the performance and training stability of machine learning algorithms that are sensitive to the scale of the input values, such as k-nearest neighbors, support vector machines, and gradient descent-based models.

- The main objective of normalization is to make different features comparable in magnitude and ensure that no single feature dominates others simply due to its scale.

- Normalization helps the optimization algorithms converge faster during the training phase by maintaining uniformity across all features.

- There are several popular methods used for data normalization, among which the most common are **min–max scaling** and **variance scaling**, also known as **standardization**.

In **min–max scaling**, each feature value is rescaled to fit within a specific range, typically between 0 and 1. This transformation is done using a formula that subtracts the minimum value of the feature from each value and then divides by the range (maximum - minimum). The formula is:

- **new_value = (value - min) / (max - min)**

- This method is particularly useful when the distribution of data is not Gaussian or when the algorithm assumes bounded input features.

- However, min–max scaling is sensitive to outliers because extreme values can skew the scale.

In **variance scaling**, also known as standardization, the features are transformed so that they have a mean of 0 and a variance of 1. This is achieved by subtracting the mean of the feature and then dividing by its standard deviation:

- **standardized_value = (value - mean) / standard_deviation**

- Standardization is often preferred when the data follows a Gaussian distribution or when the range of the data is unknown or unbounded.

- It is less affected by outliers compared to min–max scaling, making it more robust in many real-world scenarios.

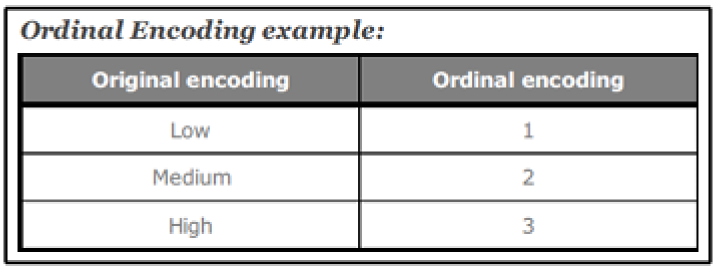
- These transformations ensure that all features contribute equally to the distance calculations and gradient updates, especially in algorithms that rely on distance metrics or assume normally distributed features.

Data normalization is often implemented using libraries like scikit-learn. For min–max scaling, the `MinMaxScaler` class is used, and for standardization, the `StandardScaler` is employed. These tools automatically handle the calculations and can transform both training and testing data consistently.

In summary, normalization is a foundational step in preparing data for machine learning, ensuring fairness and efficiency in algorithmic processing, and it significantly impacts model performance and training convergence.

MANAGING CATEGORICAL VARIABLES

Managing categorical variables is essential in machine learning because most algorithms require input data to be in numerical form. Categorical variables are features that represent discrete categories or classes, such as color, country, or profession, which do not have a natural numeric interpretation. To use these variables in models, they need to be transformed into numerical values through encoding techniques.

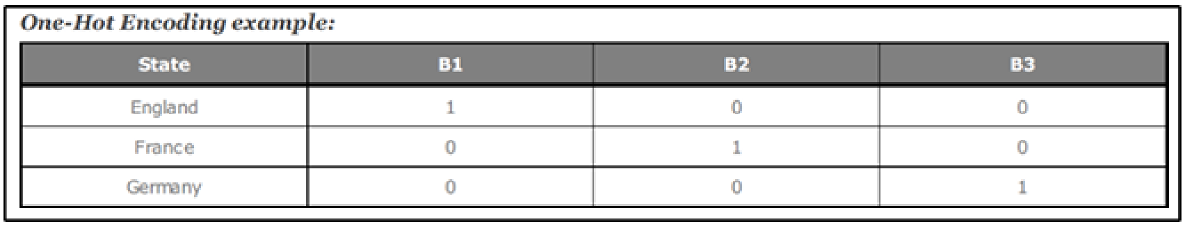
- One of the simplest and most intuitive encoding methods is **ordinal encoding**. This approach assigns a unique integer to each category. For instance, the categories "Low," "Medium," and "High" could be encoded as 0, 1, and 2 respectively.
- 

  Ordinal Encoding example:

Original encoding	Ordinal encoding
Low	1
Medium	2
High	3

- The benefit of ordinal encoding is its simplicity and low dimensionality. However, it can also introduce unintended ordinal relationships among categories that are actually nominal. This can mislead some models into interpreting a natural order that does not exist.

- The advantage and disadvantage of this encoding method is that the transformed values may be numerically ordered, even when this numerical ordering has no real meaning.
- A more widely used method is **one-hot encoding**, which avoids any assumptions about order. It creates a new binary variable (bit) for each category of the original variable. Each instance is marked with a 1 in the column corresponding to its category and 0 in all others.

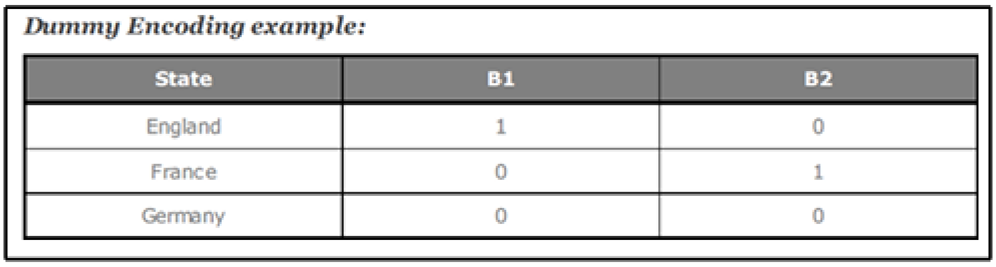


One-Hot Encoding example:

State	B1	B2	B3
England	1	0	0
France	0	1	0
Germany	0	0	1

- This method is very effective in ensuring that no ordinal relationship is inferred from the categories. However, it can lead to a large number of features when the original variable has many categories, increasing the dimensionality of the dataset.

- **Dummy encoding** is a variant of one-hot encoding. It removes one of the columns generated in one-hot encoding to avoid multicollinearity in linear models. This is because the information in the omitted category can be inferred from the others, and its presence is redundant.



Dummy Encoding example:

State	B1	B2
England	1	0
France	0	1
Germany	0	0

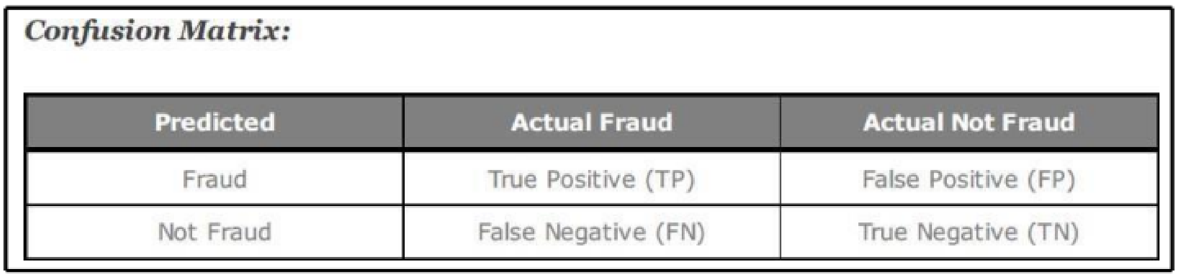
- These transformations can be efficiently implemented using libraries such as scikit-learn. The `OrdinalEncoder` and `OneHotEncoder` classes allow easy application of these techniques on both training and testing datasets.

- The choice of encoding technique depends on the nature of the data and the model being used. Algorithms like decision trees or random forests are less sensitive to feature scale or relationships and may perform well even with ordinal encoding. But for algorithms like logistic regression or support vector machines, one-hot or dummy encoding is usually more appropriate.

Effectively managing categorical variables ensures that machine learning models receive inputs in a compatible and meaningful format, which directly affects their accuracy and reliability.

DETECTORS PERFORMANCE

Evaluating the performance of a detector, particularly in cybersecurity or fraud detection, involves analyzing how well a classifier distinguishes between different classes, such as fraud and non-fraud. This is typically done using a confusion matrix, which compares the predicted results with the actual outcomes. From this matrix, several performance metrics can be derived to assess the effectiveness of the model.


Confusion Matrix:

Predicted	Actual Fraud	Actual Not Fraud
Fraud	True Positive (TP)	False Positive (FP)
Not Fraud	False Negative (FN)	True Negative (TN)

- **True Positives (TP)** represent the number of correctly predicted positive instances, such as correctly identified frauds.

- **False Positives (FP)** are the instances where the model incorrectly labels a non-fraud case as fraud.

- **True Negatives (TN)** are non-fraud cases correctly identified.

- **False Negatives (FN)** are fraud cases that the model fails to detect.

Based on these values, various evaluation metrics can be computed:

- **Sensitivity**, also known as Recall or Hit Rate or True Positive Rate (TPR), is calculated as TP / (TP + FN). It shows the proportion of actual positives correctly identified and is crucial in contexts like fraud detection where missing a fraud can be costly.

- **Specificity** is calculated as TN / (TN + FP). It measures the proportion of actual negatives correctly identified, indicating how well the model avoids false alarms.

- **False Positive Rate (FPR)** is calculated as FP / (FP + TN), or equivalently, 1 – Specificity. It indicates the proportion of non-fraud cases incorrectly flagged as fraud.

- **Accuracy** is calculated as (TP + TN) / (TP + FP + FN + TN). It represents the overall correctness of the model but may be misleading in imbalanced datasets.

- **Classification Error** is the complement of accuracy, calculated as (FP + FN) / (TP + FP + FN + TN). It reflects the rate of incorrect predictions.

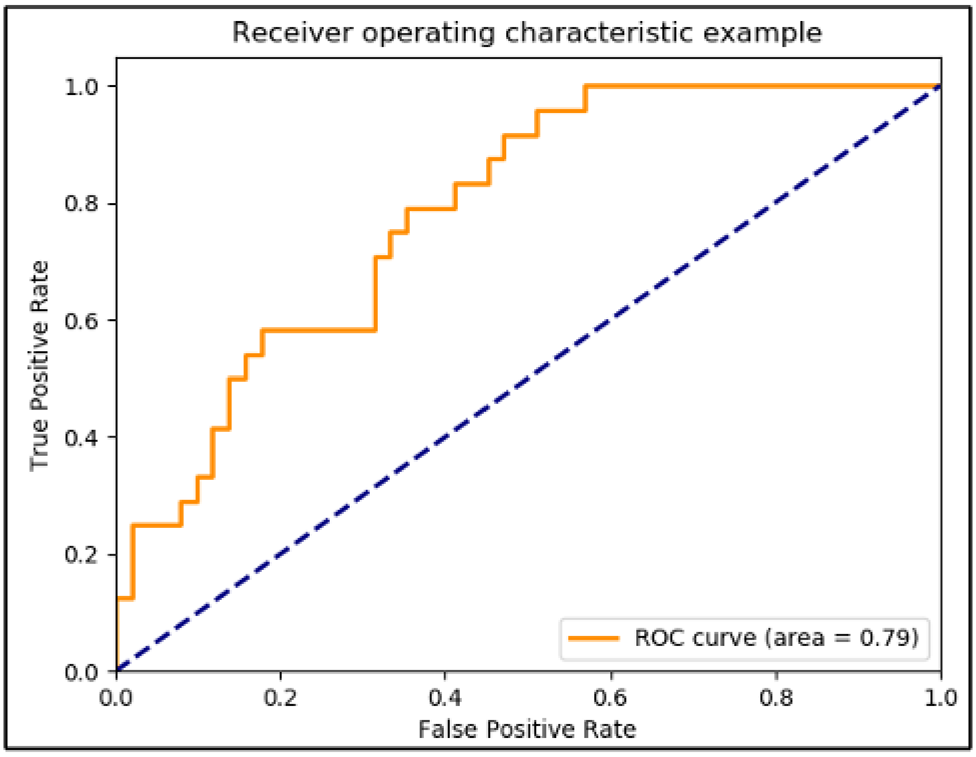
- **Precision** is calculated as TP / (TP + FP). It indicates how many of the instances predicted as positive are actually positive, which is important when the cost of false positives is high.

- **F1-Score** is the harmonic mean of Precision and Recall: 2 * (Precision * Recall) / (Precision + Recall). It balances the trade-off between precision and recall and is especially useful when the class distribution is uneven.

These metrics help in selecting and fine-tuning models, comparing their performance across different scenarios, and ensuring they operate effectively in real-world environments where the cost of errors varies depending on the application.

ROC CURVE AND AUC MEASURE

The ROC curve, or **Receiver Operating Characteristic curve**, is a graphical representation used to evaluate the performance of binary classifiers. It illustrates the trade-off between sensitivity (true positive rate) and specificity (false positive rate) across different classification thresholds.



- On the ROC plot, the x-axis represents the **False Positive Rate (FPR)**, which is calculated as FP / (FP + TN), and the y-axis represents the **True Positive Rate (TPR)**, also known as sensitivity or recall, which is calculated as TP / (TP + FN).

- Each point on the ROC curve corresponds to a different threshold used to classify a prediction as positive or negative. As the threshold varies, the values of TPR and FPR change, plotting out the curve.

- A classifier that perfectly distinguishes between the two classes will have a point in the top-left corner of the plot (FPR = 0, TPR = 1). This means it has no false positives and correctly detects all true positives.

- The diagonal line from (0,0) to (1,1) represents a random classifier. Any model that performs worse than this line is considered less effective than random guessing.

- The further the ROC curve is from the diagonal and the closer it approaches the top-left corner, the better the performance of the classifier.

The **AUC**, or **Area Under the Curve**, is a single scalar value that quantifies the overall ability of the model to discriminate between the classes, based on the ROC curve.

- The AUC value ranges from 0 to 1.
- AUC = 1 implies a perfect classifier that separates the classes without error.
- AUC = 0.5 indicates no discrimination ability—equivalent to random guessing.
- AUC < 0.5 implies that the model is predicting in reverse; swapping the labels would improve its performance.

The AUC has an important probabilistic interpretation. It can be seen as the probability that a randomly chosen positive instance is ranked higher (i.e., given a higher score or probability) than a randomly chosen negative instance by the model.

- High AUC values suggest that the model is robust and effective across various threshold values, not just one specific cutoff.
- AUC is especially useful in imbalanced datasets, where accuracy can be misleading due to a large number of negatives overpowering the positives.

To correctly evaluate the quality of the estimated probabilities of the individual classifiers, we can use the **Brier score** (BS), which measures the average of the differences between the estimated probabilities and the actual values. Here is the BS formula:

$$BS = \sum (P_i - \varphi_i)^2$$

Here, Pi is the estimated probability for observation i, and is a binary estimator (that assumes values of 0 or 1) for the actual value, i. Also, the value of BS falls in the interval between 0 and 1, but, unlike the AUC, smaller values of BS (that is, BS values closer to 0) correspond to more accurate probability estimates.

SPLIT DATA INTO TRAINING AND TESTING SETS

To evaluate the learning effectiveness of machine learning algorithms, it's essential to test their performance on data they haven't seen before. One common technique to do this is to split the dataset into two parts: a **training set** and a **testing set**. The training set is used to train the model, while the testing set is used to assess its performance. Typically, 70–80% of the data is used for training and the remaining 20–30% for testing.

- This splitting can be done using libraries like scikit-learn with the `train_test_split()` method, where the `test_size` parameter controls the portion reserved for testing.
- Although this approach is simple, the way data is split can significantly affect the model's learning outcome. If important data points are left out from the training set, or if the training data is not representative of the real-world scenario, the model may not perform well.

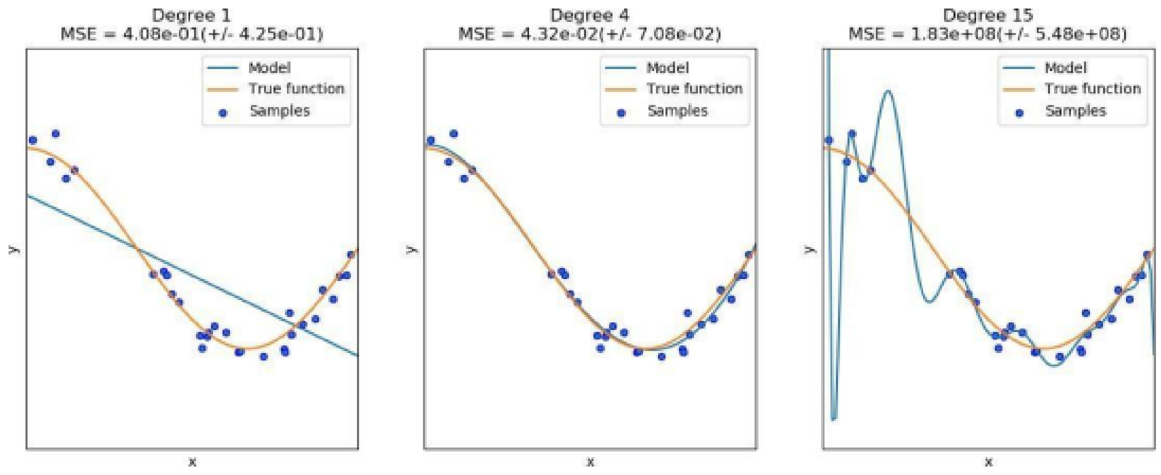
When a model is trained on sample data, it attempts to generalize its predictions to unseen data. However, **algorithm generalization error** always exists, representing the difference between the predicted and actual values on new data.

Generalization Error = Bias + Variance + Noise

- The generalization error is typically broken down into three components: **Bias**, **Variance**, and **Noise**.
    - **Bias** refers to the error due to overly simplistic assumptions in the model.
    - **Variance** is the error caused by sensitivity to small fluctuations in the training data.

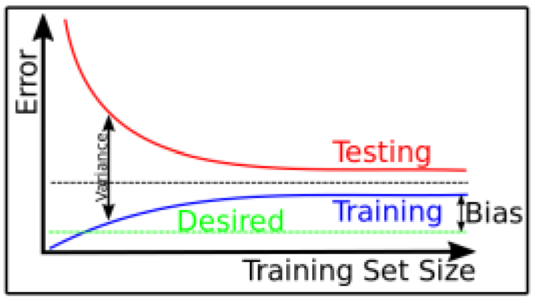
    - **Noise** is the part of the error that comes from the randomness or inherent variability in the data and cannot be reduced.

- Models with high bias underfit the data, failing to capture underlying patterns.
- Models with high variance overfit the data, capturing noise along with the signal, leading to poor performance on test data.
- A balanced model maintains low bias and low variance to achieve low generalization error.



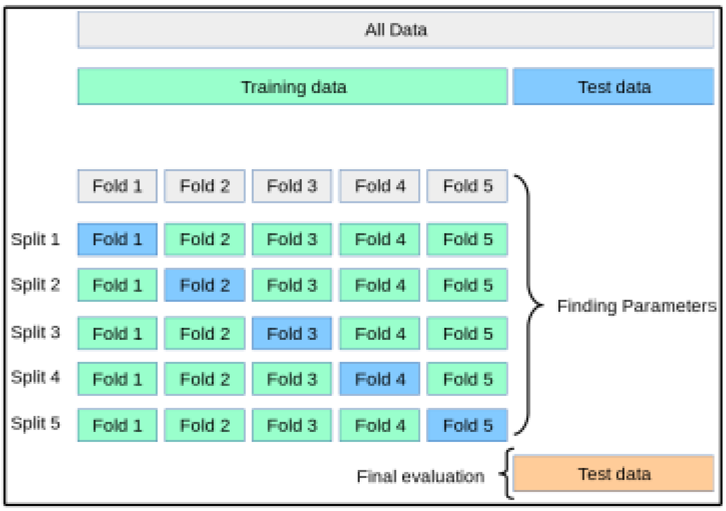
-
- A method to reduce the Variance associated with the complexity of the algorithm involves increasing the amount of data constituting the training dataset; however, it is not always easy to distinguish which component (the Bias or the Variance) assumes greater importance in the determination of the generalization error.

Algorithm learning curves are used to diagnose whether a model is suffering from high bias or high variance. A learning curve plots the training and testing accuracy (or error) against the size of the training dataset.



- If both the training and testing scores are low and converge, it indicates high bias, and increasing the model complexity may help.
- If the training score is high but the testing score is low, it shows high variance, and increasing the size of the training data can reduce the variance.
- If neither curve improves or converges, it may require tuning both model complexity and training data size.

Instead of relying on a single split, **cross validation** is used to provide a more reliable estimate of a model's performance. In cross validation, the data is divided into multiple subsets or folds.



- The model is trained on all folds except one, which is reserved for testing. This process is repeated so that each fold is used once as a test set.
- This ensures that the entire dataset is used for both training and testing at different points, leading to more stable and robust evaluation results.

The most common form is **k-fold cross validation**, where the data is split into *k* equally sized folds.

- The model is trained *k* times, each time using a different fold for testing and the remaining *k-1* folds for training.
- The final evaluation metric is the average of the metrics obtained from each fold.

Advantages of k-fold cross validation:

- Every observation is used for both training and testing, making the evaluation more reliable.
- It reduces bias in performance estimates compared to a single train-test split.
- It enables all the available data to be used, both for training and testing goals
- The specific composition of the individual folds is irrelevant since each fold is used at most once for training, and once for testing
- Increasing the value of *k* results in more training data being used in each iteration, which helps reduce the bias.

Disadvantages of k-fold cross validation:

- It assumes that the order of the data is irrelevant. This assumption may not hold true for time series data where sequence matters.

- It is computationally expensive, especially with large datasets or complex models, since the training process is repeated *k* times.

- If the dataset is not shuffled properly before splitting, the folds may end up being unrepresentative of the overall data distribution.

In scenarios like time series analysis, standard k-fold is not suitable, and alternative approaches that preserve the sequence of the data should be used.

TESTING FOR DATA AND MODEL QUALITY

Testing for data and model quality is a critical phase in the development of any machine learning solution, especially in sensitive fields like cybersecurity. Ensuring high-quality data and models guarantees reliable, robust, and effective AI systems. This involves examining the data for completeness, correctness, and balance, as well as evaluating the model for performance, generalizability, and optimal configuration.

To begin with, **assessing data quality** involves verifying the accuracy, consistency, and completeness of the dataset. Poor quality data can mislead learning algorithms, resulting in models that make incorrect or biased decisions.

- One common issue is **biased datasets**. This bias often arises from the way data is collected, known as selection bias. For example, if malware detectors are trained using data collected only from honeypots inside a corporate perimeter, the data may reflect only certain attack types specific to that environment. Such datasets will not generalize well to the broader range of threats found in the wild. This can lead to exclusion bias, where important categories of data are missing entirely.

- To reduce dataset bias, the scope of the problem should be clearly defined. The dataset should focus on the specific threats or anomalies the system is intended to detect. This helps ensure that training data reflects the real use-case accurately.

Another challenge is **unbalanced and mislabeled datasets**. In many real-world scenarios, such as credit card fraud detection, the number of legitimate transactions far outweighs the number of fraudulent ones.

- An unbalanced dataset causes the model to favor the majority class, often failing to detect rare but critical instances from the minority class.

- One effective way to address this is using resampling techniques like the Synthetic Minority Over-sampling Technique (SMOTE), which generates synthetic examples of the minority class to balance the dataset.

- Mislabeled data, on the other hand, results in faulty learning and misclassification. This may happen due to human error or flawed labeling processes, and it requires careful review or automated anomaly checks to detect and correct.

Missing values in the dataset are another frequent problem. They occur when some fields in the data are left blank or contain null entries, which can disrupt training and lead to errors.

- These can be addressed in several ways:
    - Rows or columns containing missing values can be dropped.
    - Null entries can be filled with default values (such as zero), or estimated using statistical methods like mean, median, or mode.
    - More advanced methods involve imputation techniques, such as univariate imputation using the `SimpleImputer` in scikit-learn which allows you to replace null values with a constant value, or with a positional statistics metric, such as mean, median, or mode, which is calculated on the remaining values of the column that contains the null value, or multivariate imputation using the `IterativeImputer`, which estimates missing values based on other features in the dataset.

- Each method has trade-offs. Dropping data may lose useful information, while imputation can introduce bias if done improperly.

Once the dataset is cleaned, the focus shifts to **assessing model quality**. A good model should not only perform well on the training data but also generalize well to new, unseen data.

- One of the key steps in this process is **fine-tuning hyperparameters**. These are external settings of a model that are not learned during training, such as the learning rate, number of layers, or the number of trees in a random forest.
- Hyperparameter tuning is performed by testing different combinations of values and selecting the set that yields the best performance, often based on validation data.
- Techniques like grid search and randomized search are commonly used to automate this process.

To reliably measure how well different hyperparameter settings or models generalize, **model optimization with cross validation** is used. Rather than relying on a single train-test split, cross validation evaluates the model across multiple data partitions.

- In **k-fold cross validation**, the dataset is split into *k* folds. The model is trained on *k-1* folds and tested on the remaining fold. This process is repeated *k* times, with each fold used once as the test set.
- The average performance across all folds gives a more accurate estimate of how the model is likely to perform in real-world scenarios.
- Cross validation not only helps in assessing the true performance of the model but also aids in avoiding overfitting, as the model is repeatedly tested on different data subsets.

By systematically testing for both data and model quality, including identifying and fixing biased or incomplete data, tuning hyperparameters, and using robust validation strategies, machine learning practitioners can build models that are accurate, fair, and effective across diverse real-world scenarios.

ENSURING SECURITY AND RELIABILITY

Ensuring the **security and reliability** of AI systems, particularly in cybersecurity and critical infrastructure contexts, is fundamental for maintaining trust, protecting assets, and guaranteeing uninterrupted services. This goes beyond just making the models accurate—it requires a full-spectrum strategy involving performance, scalability, resilience, availability, confidentiality, and privacy protections.

Therefore, ensuring security and reliability of AI-empowered solutions translates into the following:

- Ensuring performance and scalability
- Ensuring resilience and availability
- Ensuring confidentiality and privacy

To **ensure performance and scalability**, AI systems must be built to handle increasing loads without degrading in speed or effectiveness. Performance relates to the system's ability to process requests or data quickly and accurately, while scalability is about expanding this capability as demand grows.

- Performance bottlenecks can occur when AI models are too complex or when hardware resources are inadequate. To address this, systems should be optimized using efficient algorithms, model pruning, quantization, or using lighter architectures suitable for deployment in real-time environments.
- Preprocessing pipelines should be streamlined to reduce latency in data input and output.
- Hardware acceleration using GPUs or TPUs can dramatically increase processing speed, especially for deep learning models.
- As the system scales to handle more users or more data, cloud-based infrastructure can be employed. Cloud platforms offer elastic computing, where resources can be scaled up or down automatically depending on the load.
- Load balancing ensures traffic is evenly distributed across multiple servers, preventing any one server from being overwhelmed.

To **ensure resilience and availability**, systems must be prepared for failures and disruptions. Resilience is the ability to recover from faults and continue operating, while availability is about minimizing downtime.

- Redundancy is key—critical components should have backups so that a failure in one area does not bring down the whole system. This includes backup servers, databases, and communication channels.
- Failover mechanisms automatically reroute processes to backup systems if the primary system fails.
- Continuous monitoring tools should be integrated to track system health and performance. These tools can alert administrators in real-time or trigger automated responses to detected issues.
- Systems should be tested through chaos engineering or stress tests, where intentional disruptions are introduced to verify the system's ability to recover gracefully.
- Availability can be enhanced by deploying services across multiple geographic regions. In cloud environments, this is done using multi-region or

multi-zone architectures.

To **ensure confidentiality and privacy**, systems must protect user data from unauthorized access and prevent sensitive information from being leaked through model behavior or external threats.

- Data encryption is fundamental. All data should be encrypted at rest (in databases and storage) and in transit (during communication between systems or over the internet).
- Access control policies must enforce strict permissions, allowing only authorized users or services to access sensitive data. Role-based access control (RBAC) can help manage this securely.
- Secure authentication protocols such as OAuth2 and multi-factor authentication (MFA) reduce the risk of unauthorized access.
- To protect privacy, AI systems should adopt **privacy-preserving machine learning** techniques:

    - **Federated learning** allows model training on user devices or edge nodes without transferring data to a central server. Only model updates are shared, reducing exposure of personal data.
    - **Differential privacy** introduces controlled noise into datasets or training processes, ensuring that individual data points cannot be reconstructed from model outputs.
    - **Homomorphic encryption** allows computation on encrypted data, though it is computationally intensive and still under active research for practical use.

In addition to these core aspects, organizations should implement **security best practices across the AI lifecycle**:

- Regular code and model audits to detect vulnerabilities or data leakage risks.
- Logging and audit trails to monitor access to data and AI model interactions.
- Patch management to ensure software libraries and frameworks used in the system are updated and secure.
- Third-party risk assessments when using external tools, datasets, or APIs.

All of these practices together contribute to creating an AI system that is not only functional and accurate but also robust, trustworthy, and capable of operating safely and efficiently in dynamic and potentially adversarial environments. In critical sectors like healthcare, finance, and cybersecurity, the importance of building secure, scalable, and privacy-aware AI systems cannot be overstated.