SCIENTIFIC TEST & ANALYSIS TECHNIQUES
CENTER OF EXCELLENCE

Cross-Validation for Machine Learning Models

October 2024

Corinne Stafford, Ctr
Rohit Pai, Ctr

To develop and apply independent, tailored Scientific Test & Analysis Techniques solutions to Test and Evaluation that deliver insight to inform better decisions.

About this Publication:
This work was conducted by the Scientific Test & Analysis Techniques Center of Excellence under contract FA8075-18-D-0002, Task FA8075-21-F-0074.

For more information:
Visit, www.AFIT.edu/STAT
Email, AFIT.ENS.STATCOE@us.af.mil
Call, 937-255-3636 x4736

Technical Reviewers:
Joseph Lazarus
Anthony Sgambellone

Scientific Test & Analysis Techniques Center of Excellence
2950 Hobson Way
Wright-Patterson Air Force Base, Ohio

Version: 1, FY25

Abstract

Cross-validation is a widely used technique for training and validating machine learning models. This paper covers key concepts for machine learning model training and validation and provides an overview of common cross-validation methods: leave-one-out, leave-p-Out, k-fold, stratified k-fold, and repeated k-fold. To measure model performance or validity, many metrics are discussed, covering metrics for both classification and regression problems. These metrics enable model comparison between different types of models and models with different hyperparameters. A toy problem is presented to demonstrate the cross-validation procedure with a real data set.

Keywords: machine learning, modeling & simulation, cross-validation, validation

Table of Contents

Introduction

Machine Learning (ML) is a powerful tool that is increasingly used in both Department of Defense (DOD) systems and test and evaluation (T&E) of DOD systems. ML models may be built into DOD systems (e.g., enemy aircraft classification) or they may be part of a modeling and simulation (M&S) suite used to model system behavior as a system is developed and tested. In both applications, the ML models must be trustworthy to be assured that the system performs as required. For models to be trustworthy, they must be validated.

Validation is "the process of determining the degree to which a model or simulation and its associated data are an accurate representation of the real world from the perspective of the intended uses of the model" (US Department of Defense, 2018). Many statistical methods exist for validation, such as those described by the Institute for Defense Analysis "Handbook on Statistical Design and Analysis Techniques for Modeling & Simulation Validation" (Wojton et al., 2019). However, these methods do not explicitly account for ML models, which are trained to fit data. With access to the training data, cross-validation is a popular method that can be applied.

Cross-validation is a type of validation technique for supervised ML models that estimates how well a model will perform after it is trained. Supervised models are trained using labeled datasets to be able to make future predictions based on patterns found during training. Cross-validation methods are often more efficient in using available data to validate a model compared to other methods since they utilize the training data when assessing model performance. This paper first gives background on validation methods for ML models, then describes several cross-validation techniques and the metrics which are used to assess validity. Lastly, a toy problem of cross-validation is demonstrated with a real data set.

Background

ML models can be used for both classification and regression tasks. The model is called a classifier when it predicts a categorical response (e.g., threat/not threat), and a 'regressor' when it predicts a continuous response (e.g., signal strength). Cross-validation can be used to assess how well the model correctly predicts results. Since cross-validation is performed during model training, it can also be used to compare different possible types of models to determine which one performs best for a given data set.

This section provides background on key concepts critical to understanding cross-validation: over fitting/underfitting and the train-validation-test split. Sgambellone (2022) provides additional background on key principles for validating data-driven models.

Overfitting and Underfitting

When training a machine learning model, one must avoid overfitting or underfitting. Overfitting occurs when the model fits too closely or exactly to the data used to train the model and fails to make good predictions for any data other than the training data. On the other hand, underfitting occurs when the model is too simple or is unable to capture the relationship between the input and output variables accurately, generating a high error rate on both training data and unseen data. To avoid underfitting and overfitting, one must strike a balance to fit a model that is complex enough to capture the relationship between variables but not so complex that it begins to fit noise in the training data.

This balance can also be understood as the bias-variance tradeoff, pictured in Figure 1. Bias is a result of model inaccuracy. Figure 1 shows that the more complex a model is made, the more

the bias on the training data can be reduced. Variance refers to the variance of model coefficients. When the model becomes so complex that it fits the noise in the training data (overfitting), the calculated model parameters themselves become subject to the noise and are said to have high variance. To strike a balance between overfitting and underfitting means to acceptably reduce bias without increasing variance. Figure 1 shows the desired balance.

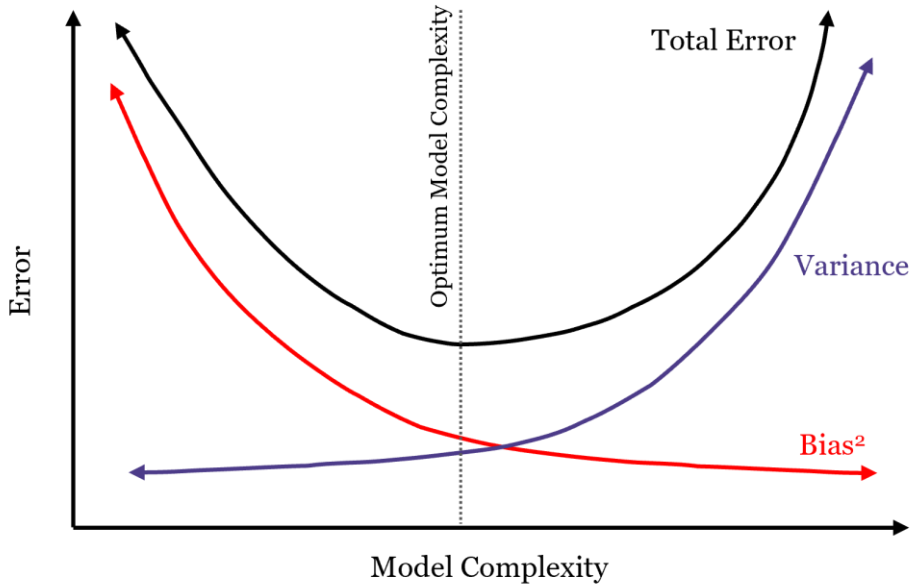

Figure 1
Bias Variance Tradeoff

Train-Validation-Test Split

Before delving into cross-validation, it is important to understand an underlying concept for validating models that are trained with data: splitting data into a training set, a validation set, and a test set. Given a dataset, it is possible to train a model on all data points; however, this fails to provide any information about the model's robustness, or ability to make predictions for data it was not trained on. One way to remedy this is by splitting the given dataset into three parts: the training set, the validation set and the test set. This is shown in Figure 2. The proportion of data in each part depends on the problem at hand. It is a best practice to include a majority proportion of the data in the training set, while the rest may be split evenly between the validation and test sets. Common splits include 60-20-20 and 50-25-25. This method is also known as the "holdout method".

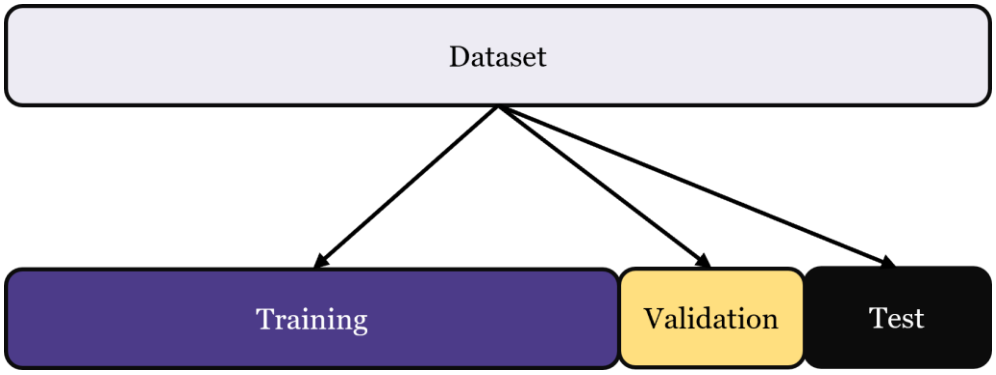

Figure 2
Train-Validation-Test Split

Training Set

The training set contains data that will be used to train the model. Different types of models each have their own algorithms for determining the best model parameters to fit the training data. For example, training a first-order linear regression model ($y = mx + b$), will output the optimal slope (m) and y-intercept (b) to minimize the error of model predictions[1]. The training data set should be the largest of the three data sets to capture as much of the behavior as possible in the model.

Validation Set

Once the model is trained, it is evaluated on the validation set. Since the validation set is not used to train the model, it provides a way to assess how the model performs on unseen data. Unlike the test set, the validation set can be used to tweak the model to perform better on unseen data. Specifically, the validation set is used to choose between models and/or choose the appropriate hyperparameters for a model. Hyperparameters differ from parameters in that they are not determined by what best fits the data. Instead, they are set by the modeler. For example, the modeler may choose what degree polynomial to fit, for example, first order (straight line), second order (quadratic), or third order (cubic). Another example of a hyperparameter is the learning rate of gradient descent when training a neural network, described further by Lazarus (2024). For different chosen hyperparameters, the resulting model will differ, and the model performance on the validation set will differ. Metrics can be calculated to evaluate this model performance, as described in the section on metrics. The modeler may select which hyperparameters result in the best performance on the validation set. Different model types (e.g., linear regression versus neural network) can also be compared to select the best model.

The validation set helps minimize overfitting. By comparing how models perform on unseen data, a model can be selected that fits both seen and unseen data well.

Test Set

Finally, the test set is used to determine generalizability of the chosen model. The test set serves as the final assessment or validation of the model against unseen data. It is critical for the test set data to be unseen and not influence the training in any way. If the test set influences training, the model may be biased to predict the test set well, and the model performance metrics will not be accurate.

[1] For a linear regression model, error is measured as the sum of squared residuals.

Disadvantages of Train-Validation-Test Split

While this approach is useful for model selection, comparison, and evaluation, it has some disadvantages. First, during the split, part of the data was arbitrarily selected as the training set and the rest was set aside the rest for validation and testing. However, depending on the specific training set selected, the created model will differ, potentially having a large impact on the model's performance. This is especially the case with smaller data sets: training the model on the largest data set possible is required to capture as much behavior as possible, and while large datasets may still contain enough data after splitting, small data sets may lose important behaviors after splitting. The validation and test data are not seen by the model during the training process, posing a possible waste of valuable, difficult-to-collect data. This motivates the need for other forms of validation for ML models that use data more efficiently.

Cross-Validation Methods

Cross-validation is a validation technique used to estimate performance of an ML model on unseen data while it is being trained. In contrast to the train-validate-test split, cross-validation trains and tunes a model using a combined training and validation set then evaluates with the test set. This combined "cross-validation set" is split in many possible ways into a training and validation set, selecting different portions of the data for training each time. For each way the data is split, a model is trained on the training set and evaluated by calculating performance metrics on the validation set. The metrics calculated for each different split can be averaged together to give an overall view of how the model performs on unseen data. Since the training data changes depending on how the data was split, cross-validation provides a more robust evaluation than a single split that only considers one possible set of training data.

Different types of models can be compared by their averaged metrics to select a model. Then, the model is retrained on the entire cross-validation set, and finally evaluated against the held-out test set. Cross-validation more efficiently uses data compared to the train-validate-test split because it eliminates the need for a separate validation set that is never used to train the model.

Below, several cross-validation methods are discussed, including leave-one-out, leave-p-out, k-fold, stratified k-fold, and repeated k-fold cross-validation.

Leave-One-Out Cross-Validation

Leave-one-out cross-validation, pictured in Figure 3, uses one piece of data as the validation set, rotating which data point serves for validation as many models are trained.

Consider a dataset with n observations. The method starts by holding out the 1st observation and training the model over the remaining n - 1 observations. The error is subsequently measured by evaluation of the model on the held-out observation. This process is repeated n times, with a different observation being left out each time. This gives n error estimates. These estimates are averaged to obtain an overall error estimate. Finally, the model may be trained on the entire dataset.

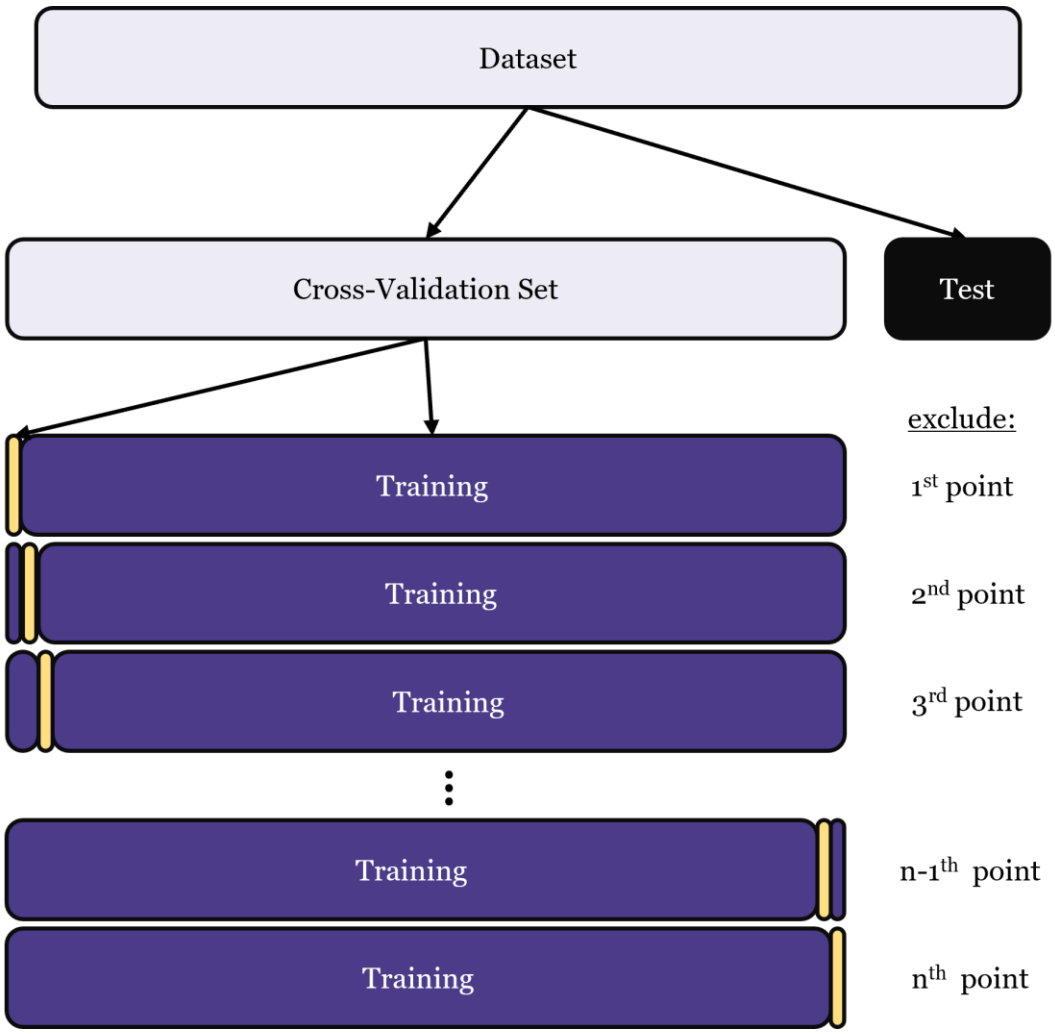

Figure 3
Leave-one-out Cross-Validation

Leave-one-out cross-validation generally results in low bias because the model is trained on almost all the data. However, leave-one-out validation can quickly become computationally straining with larger data sets, since the number of times the model is trained is equal to the size of the dataset. Thus, the leave-one-out method is better suited for smaller datasets.

Leave-p-Out Cross-Validation
A variation of leave-one-out validation is leave-p-out validation. As the name implies, each split leaves p data points out of training, where p is typically chosen to be a small number. This method exhaustively tests the model against all distinct samples of size p, so the same data point will be excluded in multiple possible splits. Thus, this method is even more computationally expensive than leave-one out since the number of possible splits grows combinatorically with the size of the data set. It should therefore only be considered for small datasets.

K-fold Cross-Validation
K-fold validation, pictured in Figure 4, is a non-exhaustive cross-validation method, and is therefore much more computationally efficient than leave-one-out or leave-p-out. Given some k

(commonly 5 or 10), the data is randomly split into k 'folds' of roughly the same size. One of the folds is set aside as the validation set, while the model is trained on the remaining k – 1 folds. An error estimate is obtained by evaluation of the model on the validation set. This process is repeated k times, with a different fold left out each time. The k error estimates are subsequently averaged to give the error.

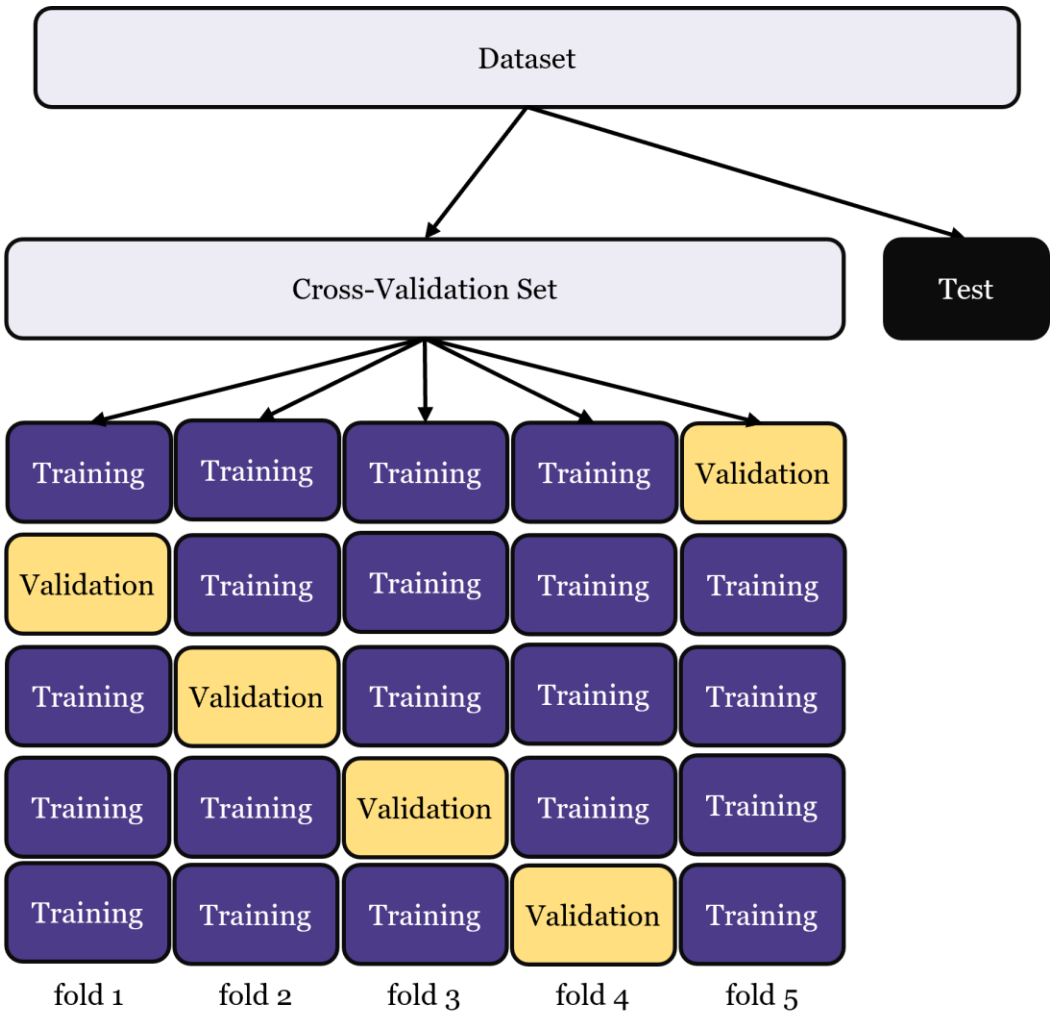

Figure 4
Five-fold (k=5) Cross-Validation

Since the number of models trained increases only with k and not with the size of the dataset, k-fold is a preferred method for large datasets. A potential downside of k-fold validation is that the random assignment of data to folds can result in unbalanced distributions of classes across folds, resulting in biased models or biased performance estimates. This can be resolved through stratification.

Stratified k-Fold Cross-Validation
Stratified sampling is a sampling method that preserves the proportions of given classes or features in the sample as in the entire data set. For example, for a detect/non-detect classifier, stratification could be used to ensure that the ratio of detect/non-detect samples in the training, validation, and test sets matches the ratio in the entire data set. When applied to k-fold

validation, each split into k folds is performed using stratified sampling as opposed to random sampling. For example, each fold would contain roughly the same ratio of detect/non-detect samples. Additionally, stratification can be applied to input features for either classifiers or regressors, e.g. radar type for a detect/non-detect model. Stratified k-fold cross-validation is the preferred method when class imbalances are present.

Repeated k-Fold Cross-Validation

Another form of k-fold cross-validation is repeated k-fold validation. As the name suggests, the process involves iterating k-fold cross-validation several times, each time with a different randomized sampling into k folds. Model metrics are averaged over all iterations. Repeated k-fold validation offers greater robustness in many cases and is helpful for understanding uncertainty due to randomly sampled folds. However, it suffers from a need for larger computational power than traditional k-fold and may not be feasible for large datasets.

Other Cross-Validation Methods

Additional methods include Monte Carlo cross-validation, which varies the split ratio in every iteration; time series cross-validation, which is used for data that follows a time trend (e.g., weather); and nested cross-validation, which uses nested (inner) sets for hyperparameter selection, and outer sets to evaluate the model. For more information see:

- Turing: Different Types of Cross-Validations in Machine Learning and Their Explanations
- Sci-kit learn: Nested versus non-nested cross-validation

Metrics

For any of the cross-validation methods described above, metrics are calculated to evaluate model validity and facilitate model selection and model tuning. Different metrics may be suitable for different models based on the type of model, e.g., classification versus numeric prediction. Also note that while these metrics are frequently used for cross-validation, they may also be used when a cross-validation methodology is not used. This section covers metrics for both classification and numeric prediction and discusses their applicability.

Classification Metrics

When evaluating how well a model classifies data points, it's useful to understand the different possible outcomes of classification.

Binary classification models can be considered to have a positive result (e.g., hit, detected, threat) and a negative result (e.g., miss, not detected, not threat). Multiclass classification models can also be considered in binary terms (class of interest or not class of interest).

Figure 5 shows all possible classification outcomes. The left (green) side represents positive observations while the right (red) side represents negative observations. Inside the circle represents positive classifications (either correct or incorrect) while outside the circle represents negative classifications. Possible outcomes are:

- true positive: occurs when a model correctly classifies an observation as positive (e.g., threat classified as threat),
- false positive: occurs when a model incorrectly classifies an observation as positive (e.g., non-threat classified as threat),
- false negative: occurs when a model incorrectly classifies an observation as negative (e.g., threat classified as non-threat),
- true negative: occurs when a model correctly classifies an observation as negative (e.g.,

non-threat classified as non-threat).

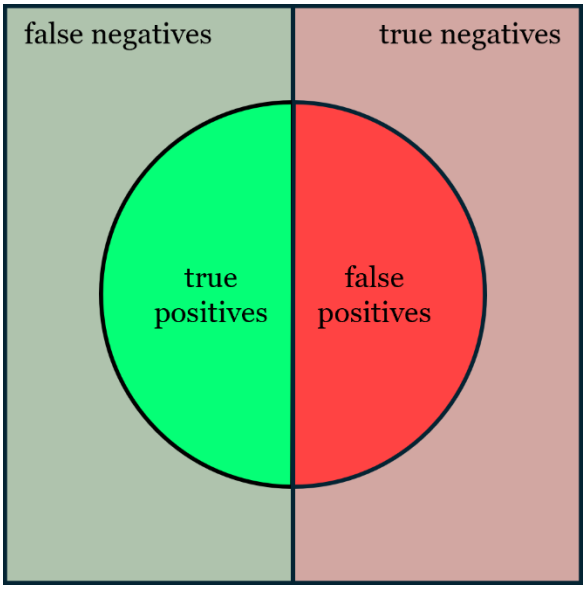

Figure 5
Possible Classification Outcomes

Each of the incorrect outcomes (false positive and false negative) may have different acceptable risk levels. For example, it may be more important to reduce the number of threats classified as non-threats instead of non-threats classified as threats if the threats will be further evaluated later. In further evaluation, non-threats may be discovered, while an observation that was never classified as a threat will be missed entirely.

Accuracy

Accuracy is a metric used to assess classification models. It is defined as the ratio of correct classifications to the total number of classifications for a dataset (Equation 1). This dataset could be the training dataset, the validation dataset, or the test dataset. To quantify validity, the accuracy should be calculated for a dataset which was not used to train the model. Obtaining poor accuracy on a validation or test set and high accuracy on a training set indicates that the model is overfitting the training data. Instead, properly fit models should show similar accuracy for the training and non-training data.

$$\text{Accuracy} = \frac{\#\text{ of correct classifications}}{\#\text{ of total classifications}} \tag{1}$$

Accuracy is a simple and straightforward metric for evaluating model goodness, however, additional metrics provide more detail.

Sensitivity and Specificity

Sensitivity and specificity are metrics used to evaluate binary classification models (e.g., to predict hit/miss, detected/not detected, threat/not threat).

Sensitivity is true positive rate, or the proportion of positives which are predicted correctly (Equation 2).

$$\text{Sensitivity} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false negatives}} \quad (2)$$

Specificity is the true negative rate, or the proportion of negatives which are predicted correctly (Equation 3).

$$\text{Specificity} = \frac{\#\text{ of true negatives}}{\#\text{ of true negatives} + \#\text{ of false positives}} \quad (3)$$

Consider, for example, a notional model that classifies whether an unknown object is a threat. The results of applying this model to a labelled validation set is shown in Table 1, which is called a confusion matrix.

		Prediction: Threat	Prediction: Not a Threat	Totals:
Actual	Threat	1	4	5
	Not a Threat	0	95	95
	Totals:	1	99	100

Table 1
Confusion Matrix of Notional Threat Classifier

The overall accuracy of this model is $96 \text{ correct} / 100 \text{ total predictions} = 0.96$. This accuracy score may seem to indicate a high performing model; however, examining the sensitivity, $1 \text{ true positives} / (1 \text{ true positives} + 4 \text{ false negatives}) = 0.2$, and the specificity, $95 \text{ true negatives} / (95 \text{ true negatives} + 0 \text{ false positives}) = 1$, reveals that while the model identifies non-threats with high probability, it fails in most cases to identify when a threat is present. Thus, both sensitivity and specificity should be examined, and both should be high to ensure a low rate of false positives and false negatives.

The decision threshold or cut-off point at which an ML model makes a classification decision influences the sensitivity and specificity of a model. For the threat classification example, lowering the threshold for what is classified as a threat may increase the number of threat classifications, increasing the sensitivity, however, this may result in more false positives, non-threats classified as threats, lowering the specificity. Setting an appropriate threshold should consider the relative risk of false negatives and false positives and the desired sensitivity and specificity.

Area Under Curve (AUC)

The AUC metric is another metric for assessing performance of a binary classifier. It is a function of the sensitivity and specificity of the model at different decision threshold settings.

Specifically, the AUC metric is the area under the model's receiving operating characteristic (ROC) curve. An example of a typical ROC curve is pictured in Figure 6. This curve is generated by plotting sensitivity versus one minus the specificity, where each point is obtained from a different threshold. The AUC can vary between zero and one, and it provides an aggregate measure of model performance across all possible decision thresholds. An AUC of one

indicates a model perfectly separates positives and negatives, and perfect sensitivity and specificity can be obtained. Any values lower than 0.5 indicate the model is no better at classifying than a coin flip.

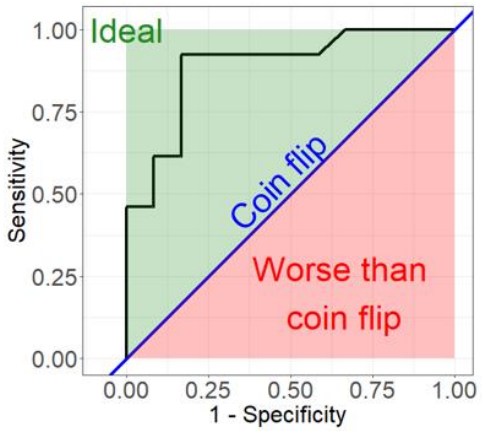

Figure 6
Example of ROC curve

In some cases, where the risk of one type of error (e.g., false positive) is very high, the AUC may not be ideal for assessing the model since it covers the entire range of possible sensitivities and specificities.

F-score

The F-score is a measure of predictive performance and is another common metric used to evaluate binary classifiers. It is a function of the precision and recall of a binary classifier, which are measures of relevance. An item is considered "relevant" if it is an actual positive observation (e.g., hit, detected, threat), and is considered "retrieved" if it is classified as positive.

Precision, also called the positive predictive value, is defined as the fraction of relevant (true positive) instances among retrieved (classified positive) instances, as in Equation 4.

$$\text{Precision} = \frac{\text{relevant instances retrieved}}{\text{all retrieved instances}} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false positives}} \quad (4)$$

Recall is equivalent to the sensitivity and is the fraction of relevant (actual positive) instances that were retrieved (true positives), as in Equation 5.

$$\text{Recall} = \frac{\text{relevant instances retrieved}}{\text{all relevant instances}} = \text{Sensitivity} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false negatives}} \quad (5)$$

Both precision and recall measure a model's ability to correctly classify positives, and both should be high for a model, where 1 is the maximum score.

The F-score is a combination of precision and recall. Variations of the F-score exist which can weight either precision or recall more strongly, but the most common variation, the F_1 score, weights precision and recall equally. Equation 6 defines the F_1 score as the harmonic mean of precision and recall, where the harmonic mean can be thought of as giving an average rate.

$$\mathrm{F}_1 = \left(\frac{\mathrm{precision}^{-1} + \mathrm{recall}^{-1}}{2}\right)^{-1} \tag{6}$$

The highest possible value of the F_1 or other F-scores is one, indicating perfect recall and precision, while the lowest is zero, indicating precision and recall are both zero.

Numeric Prediction Metrics

R-Squared

While the metrics thus far have focused on classification models, many additional metrics exist which are tailored to assess numeric prediction. A common metric used to evaluate numeric prediction is R-squared (R^2), which is the proportion of the variation in a dependent variable (output or response) which can be explained by independent variables (inputs or factors).

R-squared is defined in terms of two sums of squares formulas, the residual sum of squares, SS_{res}, and the total sum of squares, SS_{tot}, given in Equations 7 and 8, where y_i is the observed value for i^{th} data point, f_i is the predicted value for the i^{th} data point, and $\bar{y}$ is the mean of observed data.

$$\mathrm{SS}_{\mathrm{res}} = \sum_i (y_i - f_i)^2 \tag{7}$$

$$\mathrm{SS}_{\mathrm{tot}} = \sum_i (y_i - \bar{y})^2 \tag{8}$$

Then, R-squared is given in Equation 9.

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{SS_{tot}} \tag{9}$$

The maximum value of R-squared is 1, indicating that 100% of the variance in the output is explained by the model. A model which always predicts the mean will have an R-squared of zero.

Many generalizations or variations of R-squared have been created, such as for assessing classifiers, and are commonly reported for model assessment. Since there are several different definitions of R-squared, it is important to know which definition is used when understanding a reported R-squared.

Root Mean Square Error (RMSE)

RMSE is another common metric used to evaluate numeric predictions. RMSE is the square root of the average of squared errors. In this case, an error is the difference between the actual numeric value of a sample and the numeric prediction made by the model.

Equation 10 gives the RMSE, where y_i is the observed value for i^{th} data point, f_i is the predicted value for the i^{th} data point, and n is the number of data points.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2} \tag{10}$$

While RMSE and R-squared are common metrics for rating models which make numeric predictions, many other metrics such as Adjusted R-squared, Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and predicted sum of squares (PRESS), are commonly reported. These metrics are described by Burke (2020).

Toy Problem

This section walks through how cross-validation might be applied to a sample data set. The R code used is provided in the Appendix and uses the caret package, which streamlines the process of creating predictive models in R (Kuhn, 2019).

Objective and Data Set Description

This toy problem aims to show how cross-validation is beneficial for training and validating a machine learning model. Specifically, 10-fold cross-validation is compared to the holdout or train-validate-test split method.

The objective of the toy machine learning model is to classify whether an observed object is a metal cylinder (mine) or a similarly shaped rock based on sonar readings. To support this objective, the Sonar data set is used (Gorman and Sejnowski, 1988). The data set consists of 208 rows, each with 60 input variables representing the sonar-measured energy within different frequency bands and an output variable indicating the true class, whether the object is a mine ("M") or a rock ("R").

The first few rows of data are shown in Table 2. Note that several of the 60 input columns are omitted for space. For this toy example, the data set has no missing values or entry errors; however, data typically must undergo cleaning prior to modeling.

V1	V2	V3	...	V58	V59	V60	Class
0.0200	0.0371	0.0428		0.0084	0.0090	0.0032	R
0.0453	0.0523	0.0843		0.0049	0.0052	0.0044	R
0.0262	0.0582	0.1099		0.0164	0.0095	0.0078	R

Table 2
First 3 Rows Out of 208 for Sonar Data Set

As previously discussed, cross-validation is preferred over the holdout method when a data set is small. To demonstrate this, the Sonar data set will be reduced to 25% of the original size (from 208 rows to 53) by random stratified sampling, which preserves the rock/mine ratio in the larger data set. The leftover 155 rows will serve as the test set for model evaluation. Note that typically the test set would not be this large, but in this case, the larger test set shows how the model will perform after being deployed.

Model Training and Validation Approach

As stated above, two training/validation methods will be compared: train-validation-test split

(holdout) and 10-fold cross-validation. For the holdout method, the 53 rows of data are further split into training and validation sets. The data is split 80/20, with 43 training set points and 10 validation set points. The split is stratified to ensure both sets are representative of the larger population.

When training a machine learning model, many types of models are usually trained, each with tuned hyperparameters. For this toy problem, only one model type was assessed, a stochastic gradient boosted tree model implemented using the "gbm" package in R. Two hyperparameters were tuned, the number of trees and the interaction depth. The caret package chooses values for each hyperparameter and evaluates all the combinations. In this case, three possible values were tested for each hyperparameter, resulting in a total of nine combinations. The hyperparameter combination with the best performance was chosen for the final model.

In the holdout approach, models with different hyperparameter combinations were trained on the training set, and performance was evaluated on the validation set. In the cross-validation approach 10-fold validation was used, therefore models were trained excluding a different fold with each iteration, and performance metrics were averaged over all held-out folds.

To select hyperparameters, performance was evaluated with the Area under the ROC curve (AUC) metric. Recall the AUC metric quantifies how well the model separates positives and negatives, or in this case rocks and mines. The ROC metric is preferred particularly when classes are imbalanced, whereas accuracy is less useful since it does not distinguish between false negatives and false positives. In the Sonar dataset, however, no large class imbalance was present (46.6% rock, 53.4% mine).

Hyperparameter Selection

The results of hyperparameter selection are shown in Tables 3 and 4. In each table, the first two columns show the hyperparameters of the model that were varied. Additionally, the AUC for the validation set is reported (for cross-validation the average AUC across all folds is reported). The selected hyperparameter combination is highlighted in yellow, where the highest AUC with the simplest model was chosen (a lower interaction depth and fewer trees create a simpler model).

Interaction depth	Number of trees	Average AUC	Sensitivity	Specificity
1	50	0.69	0.60	0.58
1	100	0.72	0.63	0.55
1	150	0.75	0.65	0.58
2	50	0.71	0.62	0.58
2	100	0.72	0.70	0.45
2	150	0.67	0.67	0.45
3	50	0.61	0.53	0.58
3	100	0.69	0.60	0.58
3	150	0.67	0.67	0.63

Table 3
Hyperparameter Selection for Cross-Validation

Interaction depth	Number of trees	AUC	Sensitivity	Specificity
1	50	0.64	0.6	0.8
1	100	0.52	0.4	0.6
1	150	0.44	0.4	0.6
2	50	0.64	0.6	0.8
2	100	0.52	0.4	0.6
2	150	0.44	0.4	0.6
3	50	0.64	0.6	0.8
3	100	0.52	0.4	0.6
3	150	0.44	0.4	0.6

Table 4
Hyperparameter Selection for Holdout Validation

The hyperparameters selected differ between the methods, since cross-validation varies which portion of the data is held out, while the holdout method only considers one possible held-out set.

To account for uncertainty from the randomly chosen folds in k-fold cross-validation, one could perform repeated k-fold cross-validation to obtain a distribution of average AUC. These distributions could be compared between models or hyperparameter combinations to determine if one model is better, even with uncertainty.

Evaluation Against Test Set
Finally, the chosen model for each method was evaluated against the test set (155 rows). For the cross-validation method, the confusion matrix is given in Table 5, and additional evaluation metrics are provided in Table 6. For the holdout method, the confusion matrix is given in Table 7, and additional evaluation metrics are given in Table 8.

		Prediction	
		Mine	Rock
Actual	Mine	71	16
	Rock	12	56

Table 5
Confusion Matrix for Test Set using Cross-Validation Method

Metric*	Value
Accuracy	0.8194
AUC	0.8770
Sensitivity	0.8554
Specificity	0.7778
Precision	0.8161
Recall	0.8554
F1	0.8353

*Note: This table summarizes from a larger set of outputs generated by R

Table 6

Summary of Performance Metrics on Test Set using Cross-Validation Method

		Prediction	
		Mine	Rock
Actual	Mine	66	17
	Rock	17	55

Table 7

Confusion Matrix for Test Set using Holdout Method

Metric*	Value
Accuracy	0.7806
AUC	0.8484
Sensitivity	0.7952
Specificity	0.7639
Precision	0.7952
Recall	0.7952
F1	0.7952

*Note: This table summarizes from a larger set of outputs generated by R

Table 8

Summary of Performance Metrics on Test Set using Holdout Method

Comparing the performance on the test set, the model trained using cross-validation outperformed the holdout model across all metrics considered. This reflects the advantages of cross-validation. Cross-validation typically leads to more reliable hyperparameter selection, as it evaluates performance across multiple possible validation sets and uses more data for training, which often results in improved generalization and performance.

Comparing the metrics obtained for the test set versus those obtained during hyperparameter selection, the area under the ROC curve, sensitivity, and specificity are more aligned when the cross-validation method was used than when the holdout method was used. This suggests that cross-validation provides more accurate performance estimates than the holdout method.

Conclusion

Cross-validation provides a method for efficiently using data to assess model performance on non-training data and validate a model. It involves training a model multiple times on different sets of training data to assess performance on data excluded from training. While there are several different methods for performing cross-validation, k-fold cross-validation typically provides an effective approach. Metrics play a crucial role in assessing model performance and allow for model comparison. These metrics are typically tailored to the type of model, whether for classification or numeric prediction. After cross-validation, the final model is trained and evaluated against an excluded test set to validate its performance. Ultimately, employing cross-validation enables better model training through efficient uses of data and better estimates of model performance.

References

Burke, S. (2020). *The Model Building Process Part 3: Model Goodness Metrics*. Best Practice. Scientific Test & Analysis Techniques Center of Excellence. https://afit.edu/docs/Model%20Building%20Process%20Part%203%20Model%20Metrics%20Final1.pdf.

Gorman, R. P. and Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks.* 1(1), 75-89. https://doi.org/10.1016/0893-6080(88)90023-8.

Kuhn, M. (2019). The caret Package. https://topepo.github.io/caret/index.html.

Lazarus, J. (2024). *Fundamentals of Gradient Descent.* Best Practice. Scientific Test & Analysis Techniques Center of Excellence.

Sgambellone, A. (2022). *Data-Driven Model Development*. Best Practice. Scientific Test & Analysis Techniques Center of Excellence. https://www.afit.edu/docs/Data%2DDriven%20Modeling%20Best%20Practice.pdf.

US Department of Defense. (2018). Department of Defense Instruction 5000.61.

Wojton, H., Avery, K. M., Freeman, L. J., Parry, S. H., Whittier, G. S, Johnson, T. H., & Flack, A. C. (2019). Handbook on Statistical Design & Analysis Techniques for Modeling & Simulation Validation. Institute for Defense Analyses. https://www.ida.org/research-and-publications/publications/all/h/ha/handbook-on-statistical-design-and-analysis.

Appendix
R Code for Toy Problem

```
#[1] Install and load required libraries-----------------------------

#common set of packages used for data manipulation & visualization
install.packages("tidyverse")
library(tidyverse)

#machine learning methods package
install.packages("caret")
library(caret)

#Stochastic Gradient boosting package
install.packages("gbm")
library(gbm)

#example dataset package
install.packages("mlbench")
library(mlbench)

#plot ROC curve package
install.packages("pROC")
library(pROC)

#[2] Load sample data and exclude test set---------------------------

#load sonar data set
data(Sonar)

#set random seed for reproducibility
set.seed(127)

#randomly sample (stratified) down to 25% for the sake of
demonstration (53 data points)
random_sample <- createDataPartition(Sonar$Class, p=0.25, list=FALSE)
#gives indices to include
crossval_set <- Sonar[random_sample,]
test_set <- Sonar[-random_sample,]

#select validation set for comparison to train-validate split
#this creates an 80/20 percent split
set.seed(281)
val_random_sample <- createDataPartition(crossval_set$Class, p=0.80,
list=FALSE)
train_set <- crossval_set[val_random_sample,]
val_set <- crossval_set[-val_random_sample,]


#[3] Set up cross-validation-----------------------------------------
```

```
#define validation method as 10-fold cross-validation
train_control <- trainControl(method="cv",number=10,
                              ## Estimate class probabilities
                              classProbs = TRUE,
                              ## Evaluate performance using
                              ## the following function
                              summaryFunction = twoClassSummary,
                              savePredictions=TRUE)

#[4] Train model & tune hyperparameters with cross-validation---------

#fit gbm model (Stochastic gradient boosting)
set.seed(825) #set seed before training every model so that the
randomly generated folds are the same
gbm_model <- train(Class ~., data = crossval_set,
                   method = "gbm",
                   trControl = train_control,
                   verbose=FALSE,
                   metric = "ROC")
gbm_model #print results to console


#[6] Train final model & evaluate performance on test set-------------

#the model object contains the trained final model

#generate predictions for test set
test_set$predicted <- predict(gbm_model, newdata=(test_set))

#evaluate performance with confusion matrix and associated metrics
confusionMatrix(data=test_set$predicted, reference = test_set$Class)
#to obtain precision/recall/f1 use mode = "prec_recall"
confusionMatrix(data=test_set$predicted, reference = test_set$Class,
mode = "prec_recall")

#generate probabilities for each class and get ROC
M_R <- predict(gbm_model, newdata=(test_set),type="prob")
test_set_summary <- data.frame(obs = test_set$Class, pred =
test_set$predicted, M = M_R[1], R = M_R[2])
twoClassSummary(test_set_summary, lev = levels(test_set_summary$obs))

#plot ROC
plot.roc(test_set$Class,M_R$M)


#[7] Perform train-test-validate split (holdout) model training-------

train_control_h <- trainControl(method="none", classProbs = TRUE,
summaryFunction = twoClassSummary)

gbm_val_ROC <- function(i_depth,
n_tree,train_set,val_set,train_control_h){
```

```
  set.seed(825)
  gbm_model_h <- train(Class ~ ., data = train_set,
                       method = "gbm",
                       trControl = train_control_h,
                       tuneGrid = data.frame(n.trees = n_tree,
interaction.depth = i_depth, shrinkage = 0.1, n.minobsinnode=10),
                       metric = "ROC")
  val_set$predicted <- predict(gbm_model_h, newdata=(val_set))
  M_R <- predict(gbm_model_h, newdata=(val_set),type="prob")
  val_set_summary <- data.frame(obs = val_set$Class, pred =
val_set$predicted, M = M_R[1], R = M_R[2])
  tCS <- twoClassSummary(val_set_summary, lev =
levels(val_set_summary$obs))
  return(tCS)
}

i_depths <- gbm_model$results$interaction.depth
n_trees <- gbm_model$results$n.trees

ROCs <- map2_dfr(i_depths, n_trees
,~gbm_val_ROC(.x,.y,train_set,val_set,train_control_h))
cbind(i_depths,n_trees,ROCs) #report ROC for each possible
hyperparameter value
#the max ROC with the simplest model is obtained for interaction_depth
= 1, number of trees = 50


#train model with chosen hyperparameters on training set
set.seed(825)
gbm_model_h <- train(Class ~ ., data = train_set,
                     method = "gbm",
                     trControl = train_control_h,
                     tuneGrid = data.frame(n.trees = 50,
interaction.depth = 1, shrinkage = 0.1, n.minobsinnode=10),
                     metric = "ROC")
val_set$predicted <- predict(gbm_model_h, newdata=(val_set))
M_R <- predict(gbm_model_h, newdata=(val_set),type="prob")
val_set_summary <- data.frame(obs = val_set$Class, pred =
val_set$predicted, M = M_R[1], R = M_R[2])
tCS <- twoClassSummary(val_set_summary, lev =
levels(val_set_summary$obs))

#assess performance on test set
#generate predictions for test set
test_set$predicted <- predict(gbm_model_h, newdata=(test_set))

#evaluate performance with confusion matrix and associated metrics
confusionMatrix(data=test_set$predicted, reference = test_set$Class)
#to obtain precision/recall/f1 use mode = "prec_recall"
confusionMatrix(data=test_set$predicted, reference = test_set$Class,
mode = "prec_recall")
```

```
#generate probabilities for each class and get ROC
M_R <- predict(gbm_model_h, newdata=(test_set),type="prob")
test_set_summary <- data.frame(obs = test_set$Class, pred =
test_set$predicted, M = M_R[1], R = M_R[2])
twoClassSummary(test_set_summary, lev = levels(test_set_summary$obs))

#plot ROC
plot.roc(test_set$Class,M_R$M)
```