Evaluation
Sections 9.1, 9.2, 9.3

Prof. Ahmed Guessoum and Dr. Mohamed Brahimi

- When evaluating predictive models, we must ensure that the evaluation experiments are designed so that they give an accurate estimate of how the models will perform when deployed.
- The most important part of the design of an evaluation experiment for a predictive model is ensuring that the data used to evaluate the model is not the same as the data used to train the model.

Question to be answered in the Evaluation phase: Can the model generated do the job that it has been built for?

Question to be answered in the Evaluation phase: Can the model generated do the job that it has been built for?

- The purpose of evaluation is threefold:
  1. to determine which model is the most suitable for a task;
  2. to estimate how the model will perform; and
  3. to convince users that the model will meet their needs.

- The definition of best is important here since no model will ever be perfect.
- There are, though, a range of ways in which models can be incorrect, and different analytics projects will emphasize some over others
  - Medical diagnosis example: a prediction model should be very accurate in its diagnoses and, in particular, never incorrectly predict that a sick patient is healthy;
  - A model built to predict which customers would be most likely to respond to an online ad: only needs to do a slightly better than random job of selecting those customers.
- There is a spectrum of different approaches to measuring the performance of a model, and it is important to align the correct approach with a given modeling task.

Standard Approach: Measuring Misclassification Rate on a Hold-out Test Set

- **Hold-out test set**: The simplest way in which a test set can be constructed from a dataset,
- A hold-out test set is created by randomly sampling a portion of the data.

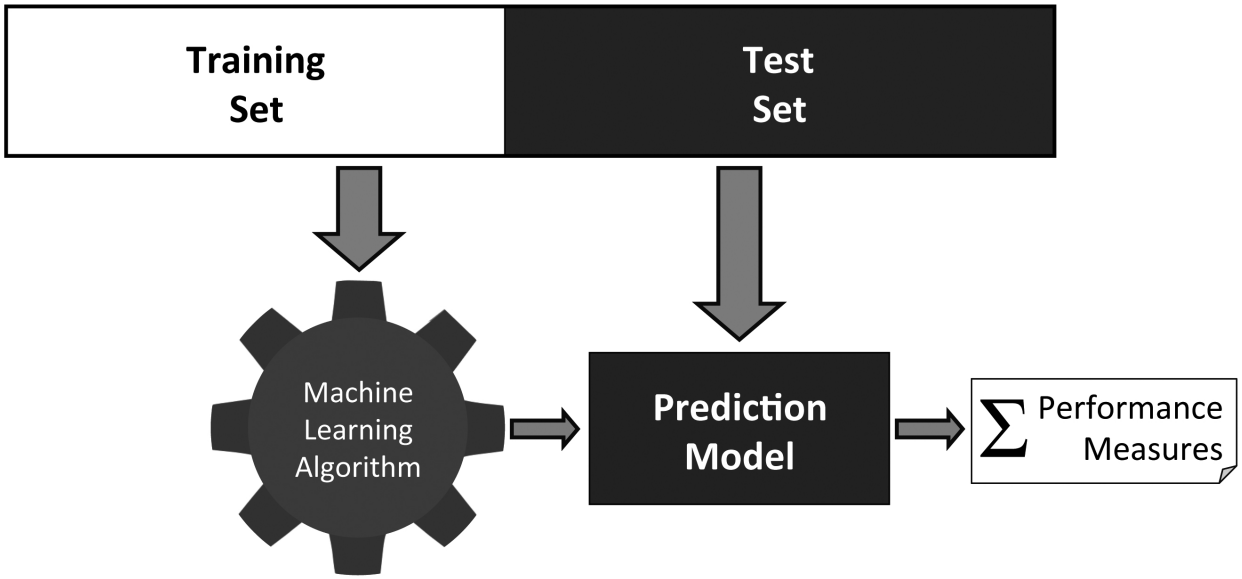


Figure: The process of building and evaluating a model using a **hold-out test set**.

- Using a hold-out test set avoids the issue of **peeking**, which arises when the performance of a model is evaluated on the same data used to train it;
- Because the data was used in the training process, the model has already *seen* this data, so it is probable that it will perform very well when evaluated on this data.
- Extreme case of this problem happens with *k-nearest neighbors* models.
- Consequently, the performance of the model on the test set:
  1. is a better measure of how the model is likely to perform when actually deployed, and
  2. shows how well the model can **generalize** beyond the instances used to train it.

Example: An email classification problem with a binary categorical target feature distinguishing between spam and ham emails.

Table: A sample test set with model predictions.

ID	Target	Pred.	Outcome	ID	Target	Pred.	Outcome
1	spam	ham	FN	11	ham	ham	TN
2	spam	ham	FN	12	spam	ham	FN
3	ham	ham	TN	13	ham	ham	TN
4	spam	spam	TP	14	ham	ham	TN
5	ham	ham	TN	15	ham	ham	TN
6	spam	spam	TP	16	ham	ham	TN
7	ham	ham	TN	17	ham	spam	FP
8	spam	spam	TP	18	spam	spam	TP
9	spam	spam	TP	19	ham	ham	TN
10	spam	spam	TP	20	ham	spam	FP

- For predictions about categorical targets, we need performance measures that capture how often the model makes correct predictions and the severity of the incorrect mistakes.

- For predictions about categorical targets, we need performance measures that capture how often the model makes correct predictions and the severity of the incorrect mistakes.
- The simplest performance measure is the **misclassification rate**.

$$\text{misclassification rate} = \frac{\text{number incorrect predictions}}{\text{total predictions}} \tag{1}$$

- For predictions about categorical targets, we need performance measures that capture how often the model makes correct predictions and the severity of the incorrect mistakes.
- The simplest performance measure is the **misclassification rate**.

$$\text{misclassification rate} = \frac{\text{number incorrect predictions}}{\text{total predictions}} \tag{1}$$

$$\text{misclassification rate} = \frac{(2+3)}{(6+9+2+3)} = 0.25$$

- For binary prediction problems there are 4 possible outcomes:
  1. **True Positive (TP)**: an instance in the test set that had a positive target feature value and that was predicted to have a positive target feature value;
  2. **True Negative (TN)**: an instance in the test set that had a negative target feature value and that was predicted to have a negative target feature value;
  3. **False Positive (FP)**: an instance in the test set that had a negative target feature value but that was predicted to have a positive target feature value;
  4. **False Negative (FN)**: an instance in the test set that had a positive target feature value but that was predicted to have a negative target feature value.

A **confusion matrix** calculates the frequencies of each possible outcome of the predictions made by a model:

Table: The structure of a confusion matrix.

		Prediction	
		positive	**negative**
Target	**positive**	*TP*	*FN*
	negative	*FP*	*TN*

- The confusion matrix can show that a model is performing well if the numbers on its diagonal are high.
- The other cells within the confusion matrix can show what kind of mistakes the model is making.

Table: A confusion matrix for the set of predictions shown in Table 1 [11].

		Prediction	
		'spam'	***'ham'***
Target	***'spam'***	6	3
	'ham'	2	9

Table: A confusion matrix for the set of predictions shown in Table 1 [11].

		Prediction	
		'spam'	***'ham'***
Target	***'spam'***	6	3
	'ham'	2	9

- We can also use the confusion matrix to investigate the kinds of mistakes that the prediction model is making.
- The kind of insight that we can get from the confusion matrix can help in trying to improve a model, as it can suggest where we should focus our work.

$$\text{misclassification accuracy} = \frac{(FP + FN)}{(TP + TN + FP + FN)} \quad (2)$$

$$\text{misclassification accuracy} = \frac{(FP + FN)}{(TP + TN + FP + FN)} \quad (2)$$

$$\text{misclassification accuracy} = \frac{(2 + 3)}{(6 + 9 + 2 + 3)} = 0.25$$

$$\text{classification accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3}$$

$$\text{classification accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3}$$

$$\text{classification accuracy} = \frac{(6 + 9)}{(6 + 9 + 2 + 3)} = 0.75$$

The most important things to take away from the preceding points:

1. It is crucial to use data to evaluate a model that has not been used to train the model.
2. The overall performance of a model can be captured in a single performance measure, for example, misclassification rate.
3. To fully understand how a model is performing, it can often be useful to look beyond a single performance measure.

Designing Evaluation Experiments

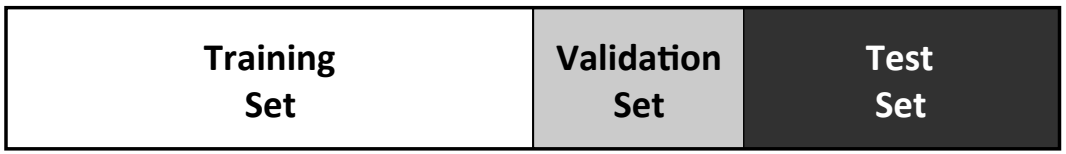
- Conflict between (1) the need to fully understand the performance of the model and (2) the need to reduce model performance to a single measure that can be used to rank models by performance.
- A set of confusion matrices gives a detailed description of how a set of models trained on a categorical prediction problem performed and can be used for a detailed comparison of performances. **However**, they cannot be ordered and so cannot be used to rank the performance of the set of models.
- such a ranking requires reduction of the information contained in the confusion matrix to a single measure (e.g. misclassification rate), hence **loss of information**.
- We introduce hereafter a selection of the most important performance measures.

- We need to select appropriate performance measures to use when evaluating trained models.
- We also need to ensure that we are using the appropriate evaluation experiment design.
- And we must ensure that we calculate the best estimate of how a prediction model will perform when actually deployed

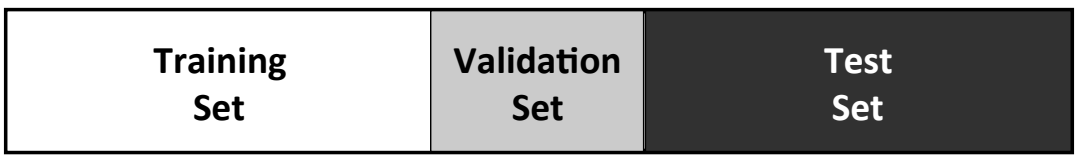
Hold-out Sampling

- A **hold-out test set** is used to evaluate the performance of a model.
- Test set not used in the training of the model.
- The performance measured on this test set should be a good indicator of how well the model will perform on future unseen data.
- In this case, the **sampling method**: we take distinct, random, non-overlapping samples from a larger dataset and use these for training and testing a prediction model.
- Hold-out sampling is probably the simplest form of sampling. It is most appropriate when the datasets are very large.

- Sometimes, hold-out sampling also uses a third sample, the **validation set**.
- The validation set is used to tune particular aspects of a model (during training).
- It is important that a separate test set still exists that can be used to evaluate the expected performance of the model.
- There are no fixed recommendations for how large the different datasets should be when hold-out sampling is used. Splits of $50 - 20 - 30$ or $40 - 20 - 40$ are common.



(a) A 50:20:30 split



(b) A 40:20:40 split

Figure: Hold-out sampling can divide the full data into training, validation, and test sets.

Hold-out Sampling

- One of the most common uses of a validation set is to avoid overfitting when using machine learning algorithms that iteratively build more and more complex models.
- We can find the point at which overfitting begins to happen by comparing the performance of a model on the training dataset and on the validation dataset.
- When the performance of the model on the validation set begins to disimprove that on the training set, this is the point at which we say overfitting has begun to occur.
- To combat overfitting, algorithms are allowed to train models beyond this point but save the model generated at each iteration.
- After the training process has completed, we find the point at which performance on the validation set began to disimprove and revert back to the model trained at that point. (Essentially the same process as that in the decision tree post-pruning.)

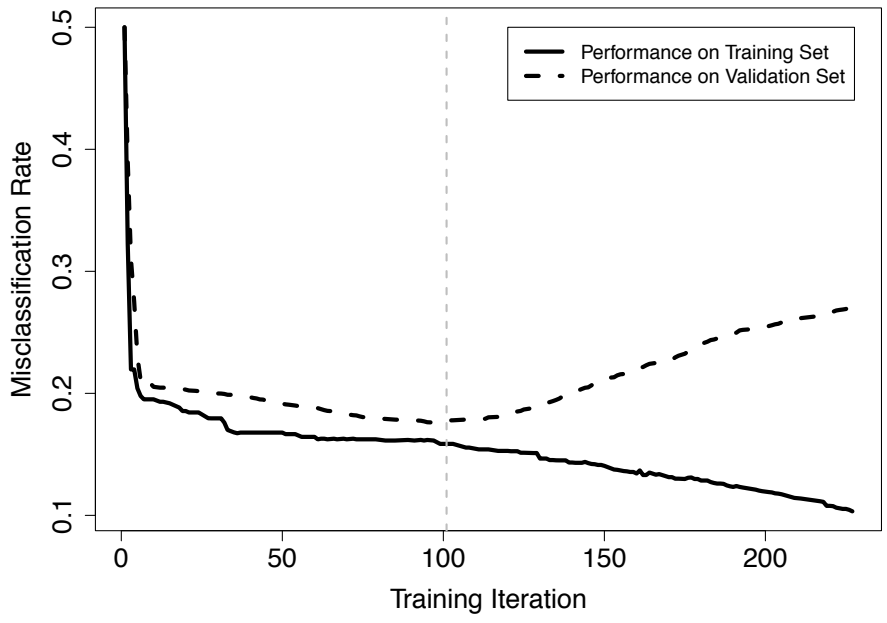


Figure: Using a validation set to avoid overfitting in iterative machine learning algorithms.

Two issues arise when using hold-out sampling:

1. Using hold-out sampling requires having enough data available to make suitably large training, test, and if required, validation sets. Making any of these partitions too small can result in a poor evaluation.
2. Performance measured using hold-out sampling can be misleading if we happen to make a **lucky split** of the data that places the difficult instances into the training set and the easy ones into the test set.

- In **k-fold cross validation** the available data is divided into k equal-sized folds (or partitions), and k separate evaluation experiments are performed.
- In the first evaluation experiment, the data in the 1st fold is used as the test set, and the data in the remaining $k - 1$ folds is used as the training set.
- Then the 2nd fold data is used as a a test set and the remainder for training; etc.
- Finally, the k sets of performance measures are aggregated to give one overall set of performance measures (**Aggregate confusion matrix**).
- k can take any value, but $k = 10$ is probably the most common variant used in practice.

k-Fold Cross Validation

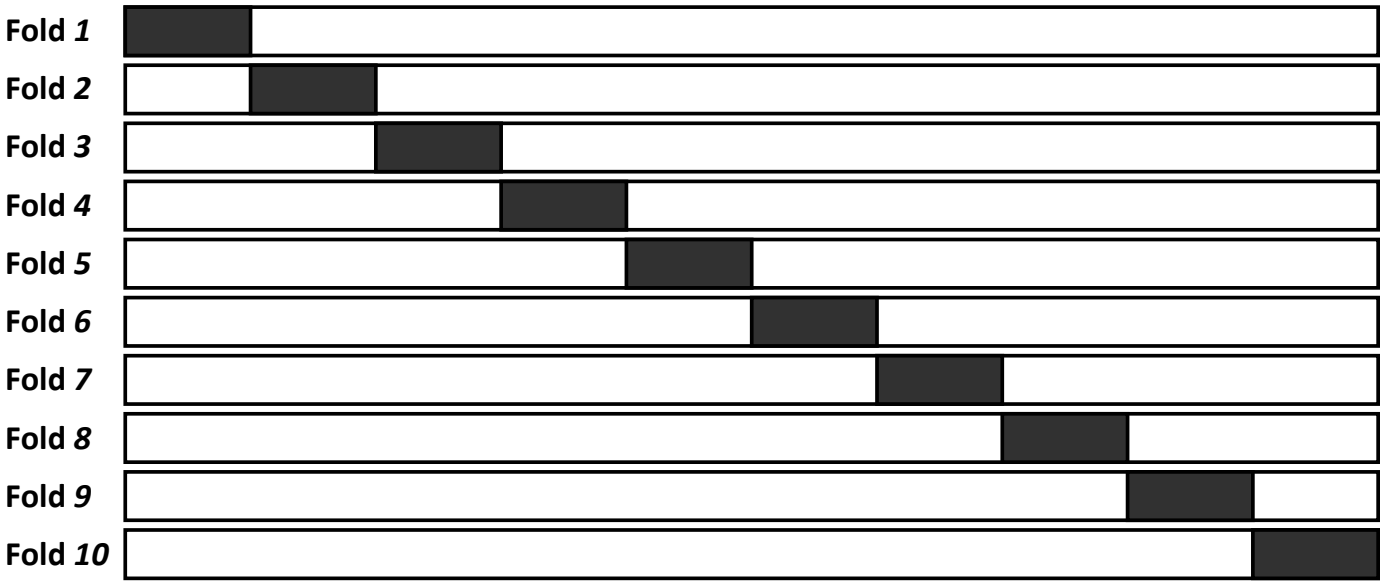


Figure: The division of data during the ***k*-fold cross validation** process. Black rectangles indicate test data, and white spaces indicate training data.

Fold	Confusion Matrix				Class Accuracy
1			Prediction *'lateral'*	Prediction *'frontal'*	81%
	Target	*'lateral'*	43	9	
		'frontal'	10	38	
2			Prediction *'lateral'*	Prediction *'frontal'*	88%
	Target	*'lateral'*	46	9	
		'frontal'	3	42	
3			Prediction *'lateral'*	Prediction *'frontal'*	82%
	Target	*'lateral'*	51	10	
		'frontal'	8	31	
4			Prediction *'lateral'*	Prediction *'frontal'*	85%
	Target	*'lateral'*	51	8	
		'frontal'	7	34	
5			Prediction *'lateral'*	Prediction *'frontal'*	84%
	Target	*'lateral'*	46	9	
		'frontal'	7	38	
Overall			Prediction *'lateral'*	Prediction *'frontal'*	84%
	Target	*'lateral'*	237	45	
		'frontal'	35	183	

- With a small dataset (introducing the possibility of a lucky split), measuring aggregate performance using a set of models gives a better estimate of post-deployment performance than measuring performance using a single model.
- After estimating the performance of a deployed model using *k-fold cross validation*, the model that will be deployed is typically trained using all of the available data.

- **Leave-one-out cross validation** is an extreme form of k-fold cross validation in which the number of folds is the same as the number of training instances.
- So each fold of the test set contains only one instance, and the training set contains the remainder of the data.
- It is useful when the amount of data available is too small to allow big enough training sets in a k-fold cross validation.
- At the conclusion of this process, a performance measure will have been calculated for every instance in the dataset.
- These performance measures are aggregated across all the folds to arrive at an overall measure of model performance.

Leave-one-out Cross Validation

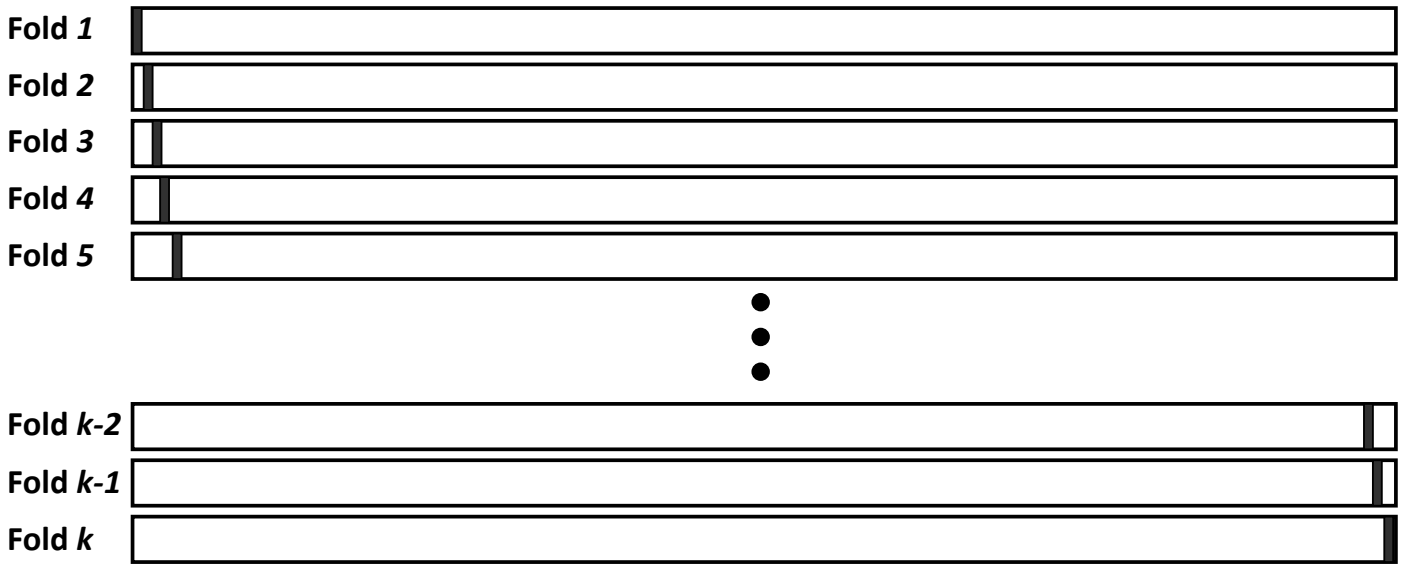


Figure: The division of data during the **leave-one-out cross validation** process. Black rectangles indicate instances in the test set, and white spaces indicate training data.

- **Bootstrapping** approaches are preferred over cross validation approaches in contexts with very small datasets (about fewer than 300 instances).
- To generate the partitions for an iteration of the *e*0 bootstrap, a random selection of *m* instances (with replacement) is taken from the full dataset to generate a test set; the remaining instances are used as the training set.
- Using the training set to train a model and the test set to evaluate it, a performance measure (or measures) is calculated for this iteration.
- *k* iterations performed, and the overall model performance is the average of the individual performance measures.
- Typically, in the *e*0 bootstrap, $k \geq 200$.

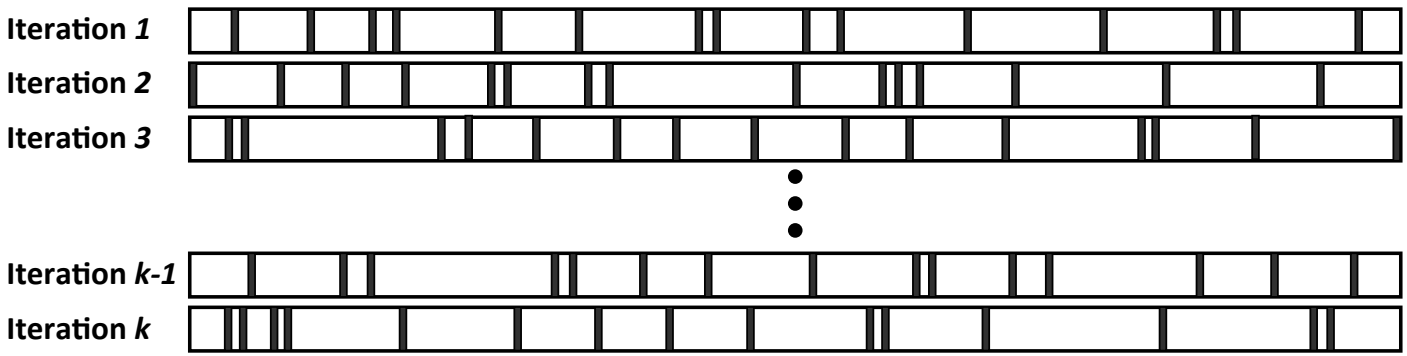


Figure: The division of data during the $\epsilon 0$ bootstrap process. Black rectangles indicate test data, and white spaces indicate training data.

Out-of-time Sampling

- So far the discussed sampling methods all rely on random sampling from a large dataset in order to create test sets.
- In some applications there is a natural structure in the data that we can take advantage of to form test sets.
- In scenarios that include a time dimension, this is often referred to as **out-of-time sampling**, because we use data from one period to build a training set and data out of another period to build a test set.
- Example: Customer churn scenario: one might use details of customer behavior from one year as a training set and details of customer behavior from a subsequent year as a test set.

- Out-of-time sampling is a form of hold-out sampling in which the sampling is done in a targeted rather than a random fashion.
- Here one should be careful to ensure that the times from which the training and test sets are taken do not introduce a bias into the evaluation process, because the two different time samples are not really representative.
- E.g., estimation of energy demand: training/test data based on weather, time of year, etc.
- It is important when choosing the periods for out-of-time sampling that the time spans are large enough to take into account any cyclical behavioral patterns or that other approaches are used to account for these.

Out-of-time Sampling

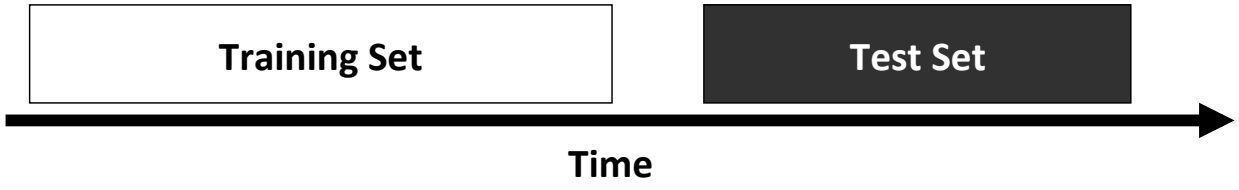


Figure: The **out-of-time sampling** process.

Performance Measures: Categorical Targets

- Confusion matrices are a convenient way to fully describe the performance of a predictive model.
- They are also the basis for a whole range of different performance measures that can highlight different aspects of the performance of a predictive model.
- The most basic of these measures are **true positive rate (TPR)**, **true negative rate (TNR)**, **false negative rate (FNR)**, and **false positive rate (FPR)**

$$\text{TPR} = \frac{TP}{(TP + FN)} \tag{4}$$

$$\text{TNR} = \frac{TN}{(TN + FP)} \tag{5}$$

$$\text{FPR} = \frac{FP}{(TN + FP)} \tag{6}$$

$$\text{FNR} = \frac{FN}{(TP + FN)} \tag{7}$$

Their values are in the range [0, 1].

For the email classification data given previously:

$$\begin{aligned} \text{TPR} &= \frac{6}{(6+3)} &= 0.667 \\ \text{TNR} &= \frac{9}{(9+2)} &= 0.818 \\ \text{FPR} &= \frac{2}{(9+2)} &= 0.182 \\ \text{FNR} &= \frac{3}{(6+3)} &= 0.333 \end{aligned}$$

For the email classification data given previously:

$$\begin{aligned} \text{TPR} &= \frac{6}{(6+3)} &= 0.667 \\ \text{TNR} &= \frac{9}{(9+2)} &= 0.818 \\ \text{FPR} &= \frac{2}{(9+2)} &= 0.182 \\ \text{FNR} &= \frac{3}{(6+3)} &= 0.333 \end{aligned}$$

These values immediately suggest that the model is better at predicting the ham level (TNR) than it is at predicting the spam level (TPR).

Precision, **Recall**, and the **F1 measure** are another frequently used set of performance measures:

$$\text{precision} = \frac{TP}{(TP + FP)} \tag{8}$$

$$\text{recall} = \frac{TP}{(TP + FN)} \tag{9}$$

- Precision tells us how confident we can be that an instance predicted to have the positive target level actually has the positive target level.
- Recall tells us how confident we can be that all the instances with the positive target level have been found by the model.

For the mail classification data,

- precision measures how often the emails marked as spam actually are spam;
- recall measures how often the spam messages in the test set were actually marked as spam

$$\text{precision} = \frac{6}{(6+2)} = 0.75$$

$$\text{recall} = \frac{6}{(6+3)} = 0.667$$

Precision and recall can be collapsed to a single performance measure known as the **F1 measure**, a useful alternative to the simpler misclassification rate.

$$F_1\text{-measure} = 2 \times \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})} \tag{10}$$

Precision and recall can be collapsed to a single performance measure known as the **F1 measure**, a useful alternative to the simpler misclassification rate.

$$\text{F}_1\text{-measure} = 2 \times \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})} \tag{10}$$

$$\begin{aligned}\text{F}_1\text{-measure} &= 2 \times \frac{\left(\frac{6}{(6+2)} \times \frac{6}{(6+3)}\right)}{\left(\frac{6}{(6+2)} + \frac{6}{(6+3)}\right)} \\ &= 0.706\end{aligned}$$

The $F1 - measure$ is the harmonic mean of precision and recall.

- Precision, recall, and the F1 measure work best in prediction problems with binary target features and place an emphasis on capturing the performance of a prediction model on the positive, or most important, level.
- They place less emphasis on the performance of the model on the negative target level.
- This is appropriate in many applications, e.g. in medical applications, where a prediction that a patient has a disease is much more important than a prediction that a patient does not.
- In many cases, however, it does not make sense to consider one target level as being more important. The **average class accuracy** performance measure can be effective in these cases.

Classification accuracy can mask poor performance.

Table: A confusion matrix for a k-NN model trained on a churn prediction problem.

		Prediction	
		'non-churn'	***'churn'***
Target	***'non-churn'***	90	0
	'churn'	9	1

Table: A confusion matrix for a naive Bayes model trained on a churn prediction problem.

		Prediction	
		'non-churn'	***'churn'***
Target	***'non-churn'***	70	20
	'churn'	2	8

- First model: 91% accuracy; 2nd model: 78% accuracy.
- The test set is imbalanced: 90 instances with the non-churn level and just 10 instances with the churn level.
- This means that the performance of the model on the non-churn level overwhelms the performance on the churn level in the accuracy calculation and illustrates how classification accuracy can be a misleading measure of model performance.

To address this issue, we can use average class accuracy instead of classification accuracy:

$$\text{average class accuracy} = \frac{1}{|levels(t)|} \sum_{l \in levels(t)} \text{recall}_l \qquad (11)$$

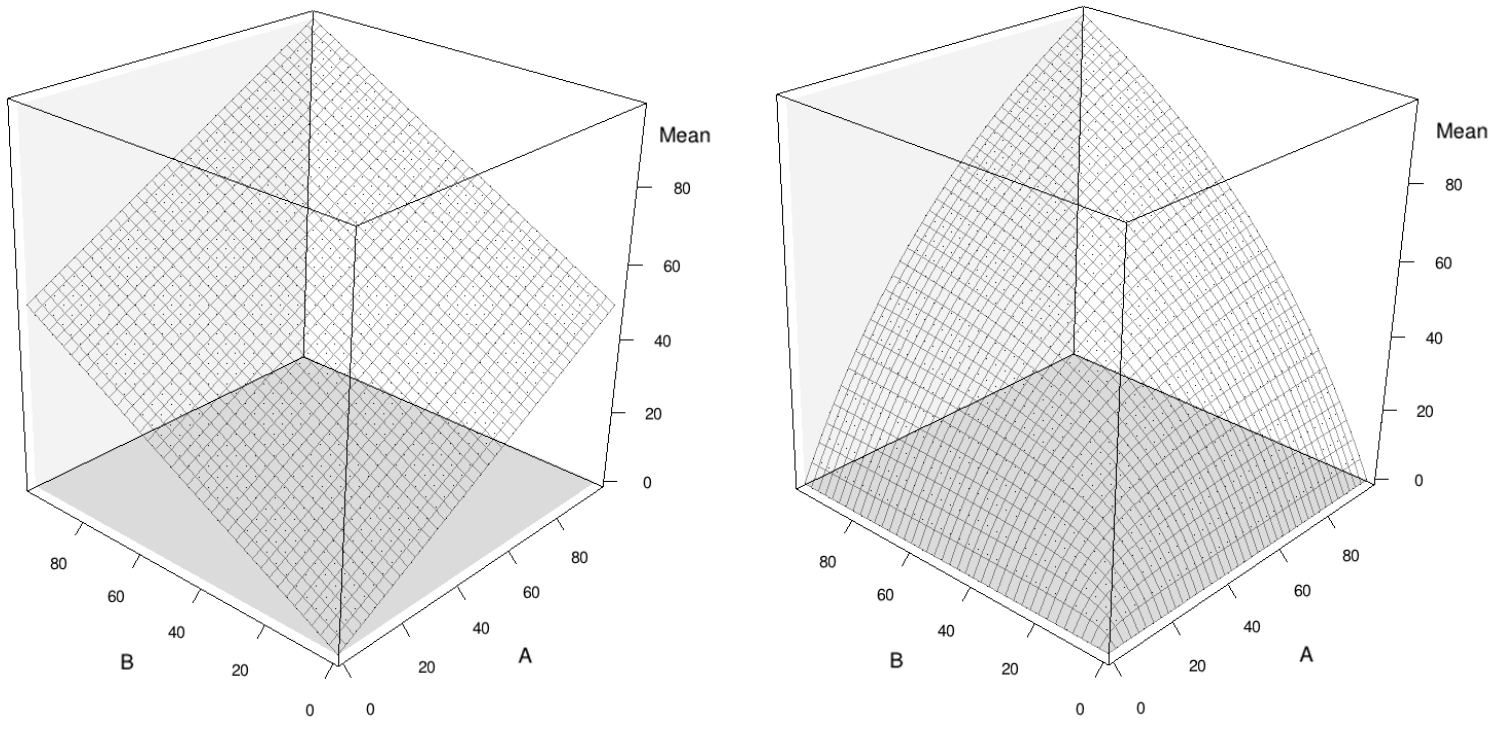
$$\text{average class accuracy}_{\text{HM}} = \frac{1}{\frac{1}{|\mathit{levels}(t)|}\sum_{l \in \mathit{levels}(t)} \frac{1}{\text{recall}_l}} \quad (12)$$

$$\frac{1}{\frac{1}{2}\left(\frac{1}{1.0}+\frac{1}{0.1}\right)}=\frac{1}{5.5}=18.2\%$$

$$\frac{1}{\frac{1}{2}\left(\frac{1}{0.778}+\frac{1}{0.800}\right)}=\frac{1}{1.268}=78.873\%$$

This would indicate that the second model is actually a better performer than the first.
This result is contrary to the conclusion drawn from classification accuracy but is more appropriate in this case due to the target level imbalance present in the data.

- Many people prefer to use a harmonic mean instead of an arithmetic mean when calculating average class accuracy.
- Arithmetic means are susceptible to influence of large outliers, which can inflate the apparent performance of a model.
- The harmonic mean, on the other hand, *emphasizes the importance of smaller values* and so can give a slightly more realistic measure of how well a model is performing.



(a) (b)

Figure: Surfaces generated by calculating (a) the **arithmetic mean** and (b) the **harmonic mean** of all combinations of features A and B that range from 0 to 100.

- It is not always correct to treat all outcomes equally
- In these cases, it is useful to take into account the cost of the different outcomes when evaluating models

Table: The structure of a **profit matrix**.

		Prediction	
		positive	**negative**
Target	**positive**	TP_{Profit}	FN_{Profit}
	negative	FP_{Profit}	TN_{Profit}

Table: The **profit matrix** for the pay-day loan credit scoring problem.

		Prediction	
		'good'	***'bad'***
Target	***'good'***	140	−140
	'bad'	−700	0

Table: (a) The confusion matrix for a k-NN model trained on the pay-day loan credit scoring problem (average class accuracy$_{HM} = 83.824\%$); (b) the confusion matrix for a decision tree model trained on the pay-day loan credit scoring problem (average class accuracy$_{HM} = 80.761\%$).

(a) k-NN model

		Prediction	
		'good'	***'bad'***
Target	***'good'***	57	3
	'bad'	10	30

(b) decision tree

		Prediction	
		'good'	***'bad'***
Target	***'good'***	43	17
	'bad'	3	37

Table: (a) Overall profit for the *k*-NN model using the profit matrix in Table 7 [63] and the **confusion matrix** in Table 8(a) [64]; (b) overall profit for the decision tree model using the profit matrix in Table 7 [63] and the **confusion matrix** in Table 8(b) [64].

(a) *k*-NN model

		Prediction	
		'good'	***'bad'***
Target	***'good'***	7 980	−420
	'bad'	−7 000	0
	Profit		**560**

(b) decision tree

		Prediction	
		'good'	***'bad'***
Target	***'good'***	6 020	−2 380
	'bad'	−2 100	0
	Profit		**1 540**

Summary