

Introduction to Machine Learning (for AI)

Cross Validation and Evaluation

Dr. Andreea Sburlea & Dr. Matias Valdenegro & Dr. Marco
Zullich

November 26, 2024

Today's Lecture

- This lecture covers the basics of cross-validation and how it relates to detecting overfitting.
- You will learn the basic concept of overfitting, how to detect it, and best practices for using cross-validation to detect overfitting.
- Additionally we cover Leakage and its importance in cross-validation.
- We also cover the science of evaluation, metrics and loss functions and their differences.

Outline

1 Cross Validation Methods

Overfitting

Cross Validation

2 Evaluation and Metrics

This Section on Evaluation

- Today we will talk about performance metrics for ML models
 - Difference between loss function and metric
 - Metrics for regression and classification
- We will see when using which type(s) of metric
- We will discuss about the relationships between different metrics

Note that this section on evaluation is a bit more detailed, it is meant for you to study and future machine learning practice. There are examples for your future study that we might skip in the lecture.

Learning Performance

The first thing is to define how to measure the performance of a learning model.

Losses

Objective function that guides learning during the optimization process. It defines the task to be learned and the quality of solutions. Usually it has to be differentiable.

Metrics

Measurements of quality that let the ML developer and **users of the model** evaluate the learning process' success. May be non-differentiable. Losses can be used as Metrics as well.

Q: Why mention non-differentiability?

Loss Functions - Regression - MAE and MSE

Mean Absolute Error (MAE)

This loss often converges to the median of the targets. Has trouble with gradients close to zero.

$$L(y, \hat{y}) = \frac{\sum_i |\hat{y}_i - y_i|}{n}$$

Mean Square Error (MSE)

This loss often converges to the mean of the targets.

$$L(y, \hat{y}) = \frac{\sum_i (\hat{y}_i - y_i)^2}{n}$$

Comparing MAE vs MSE

- MSE penalizes big errors way more than MAE (because of the square)
 - MSE is more sensitive to outliers
- MAE has a point of non-differentiability (when both y and \hat{y} have the same value); MSE is fully differentiable
- MSE retains desirable statistical properties, hence it's a common choice for linear regression
- Unit of measure of MSE is squared, while MAE is the same as the original data
 - If we do regression on length (e.g., using cm as unit of measure), then MSE will be in cm^2 , while MAE will be in cm
 - \rightarrow MSE can be difficult for human interpretation

Loss Functions - Regression - RMSE

Root Mean Square Error (RMSE)

To *mediate* between MAE and MSE, we can use the Root Mean Square Error (RMSE):

$$L(y, \hat{y}) = \sqrt{\frac{\sum_i (\hat{y}_i - y_i)^2}{n}}$$

- The loss is fully differentiable
- Is more *robust* to outliers
- Unit of measure is the same as the original data

Outliers? Outliers!

Outliers can be problematic for models trained with MSE. We should always run some checks on our data to test for the presence of outliers.

Outliers are usually not easy to detect (there are multiple methods, post-hoc checks on the residuals for linear models, RanSaC, visual checks using pair plots, *etc.*).

Outliers should not be dealt with automatically! Ask yourself a question first: *Why is there an outlier here?*

There could be **errors in measurement**

⇒ **Remove** the outlier, or use a robust loss function (Huber, ramp, *etc.*).

The outlier is a **legitimate data point**

⇒ Reason about **keeping** the outlier, ponder on the type of loss function to use.

Metrics - Regression - R^2

R^2 Score

It is a metric that measures the proportion of the variance in the labels y that is predictable from the model's predictions \hat{y} .

It answers the question «*How much (what %) of the total variation in y is explained by the variation in the regression line?*»

It can be interpreted as a measure of *goodness of fit*.

Total variation: $SD(y, \bar{y}) = \sum_i (y_i - \bar{y})^2$

Variation explained by the model: $SD(y, \hat{y}) = \sum_i (y_i - \hat{y})^2$

$$R^2(y, \hat{y}) = 1 - \frac{SD(y, \hat{y})}{SD(y, \bar{y})} \quad (1)$$

Metrics - Regression - R^2 - Examples

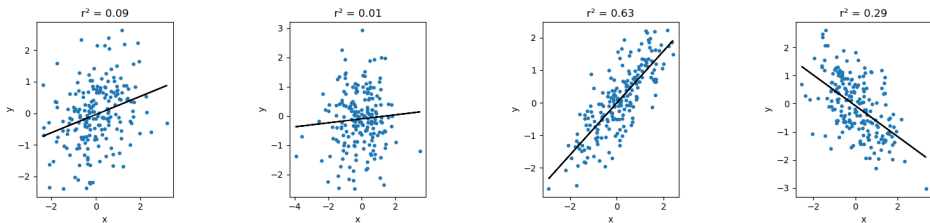


Figure: Example of R^2 attained by linear regressors fitted to various toy datasets.

Metrics - Regression - R^2 - Examples

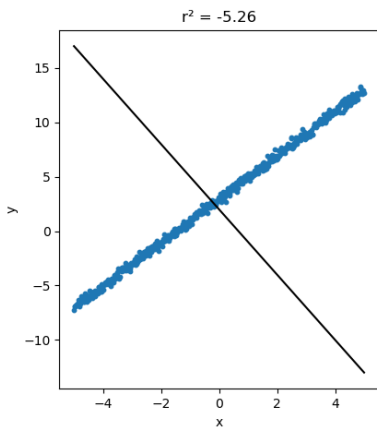
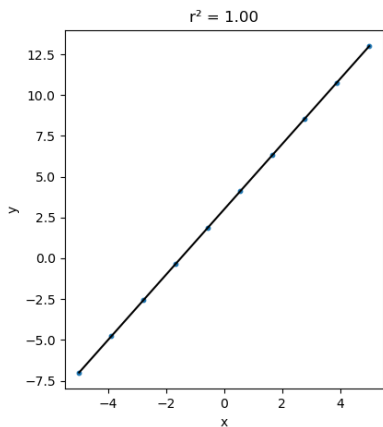


Figure: Example of R^2 attained by regressors fitted to various toy datasets.

Metrics - Regression - Adjusted R^2

R^2 Score

R^2 is usually computed on the whole dataset (no train/test split).

Adding variables usually results in a better fit.

Adding too many variables can result in overfitting.

We can introduce a *penalty term* on R^2 to account for the number of features used in the model.

$$R_{\text{adj}}^2 = 1 - \left[\frac{n-1}{n-p-1} (1 - R^2) \right] \quad (2)$$

Intuition: if we increase p (number of features), the term $\frac{n-1}{n-p-1}$ goes up, which increases $1 - R^2$, hence reducing R^2 , and n is the number of samples in your dataset.

Metrics - Regression - Correlation Coefficient

(Pearson) Correlation Coefficient

This metric measures how the variables are linearly related. It ranges between $-1 \leq \rho \leq 1$.

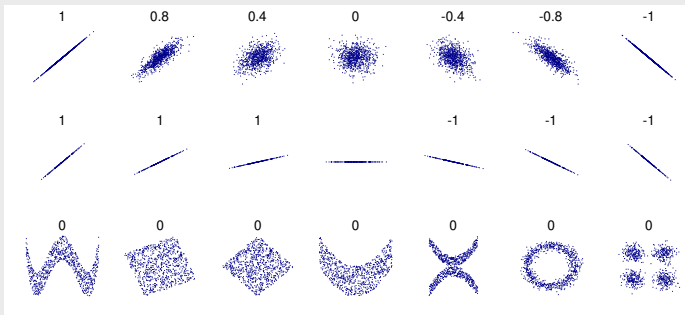
$$\rho(y, \hat{y}) = \frac{\sum_i (y_i - \mu_{y_i})(\hat{y}_i - \mu_{\hat{y}_i})}{\sqrt{n \sum_i y_i^2 - (\sum y_i)^2} \sqrt{n \sum_i \hat{y}_i^2 - (\sum \hat{y}_i)^2}}$$

Metrics - Regression - Correlation Coefficient

(Pearson) Correlation Coefficient

The correlation coefficient measure *strength* and *direction* of the linear relationship:

- Positive \Rightarrow increase in y corresponds to increase in \hat{y}
- Negative \Rightarrow increase in y corresponds to decrease in \hat{y}



Metrics for regression - summary

- MAE vs MSE vs RMSE are *absolute* metrics for regression, commonly employed as losses.
 - MAE is more robust to outliers & is good for *robust regression*
 - MSE penalizes large errors & is good for statistical properties of linear regression
 - RMSE and MAE are better for human interpretation because their values have the same unit of measure as the original data
 - There are other variants of MAE/MSE (e.g., Huber Loss), which can be helpful for mitigating the effect of outliers in regression problems.

Metrics for regression - summary

- R^2 is a *pseudo-relative* metric for regression and measures the variance in the target *explained* by the model. It can be interpreted as *goodness of fit*.
 - Linear regression models usually achieve $R^2 \in [0, 1]$, thus it can be viewed as relative, which makes it more human-interpretable
- Adding features, which can lead to overfitting, always increases R^2 . R^2_{adj} incorporates a penalty term that decreases R^2 when adding features.
 - R^2 is a good metric for getting an idea on the *performance* of a single model, but it's generally not the best for comparing different models.
 - When comparing linear regression models (especially if they have different number of features), it is better to use R^2_{adj} .

Metrics - Classification

Accuracy

A common **classification only** metric that is easily interpretable by humans.

$$\text{Acc}(y, \hat{y}) = n^{-1} \sum_i 1[y_i = \hat{y}_i]$$

Where $1[x]$ is the indicator function, returning 1 if x is true, and 0 if x is false.

Its range is $[0, 1]$, with the best value being 1. If multiplied by 100 it can be interpreted as a percentage.

Top-k Accuracy

Accuracy computed as considering any of the top k class predictions as correct. Typically used with classifiers that can rank their class predictions, and for tasks with a large number of classes.

Accuracy - Shortcomings in binary classification

Accuracy tells you *how many* data points were correctly classified.

It does not make any distinction on the *nature* of the errors. In binary classification, we usually want to

distinguish between the two possible errors:

False Negatives (FN): items belonging to the *positive* class classified in the *negative* class.

Example: credit card *fraud* transaction classified as *legitimate*.

False Positives (FP): items belonging to the *negative* class classified in the *positive* class.

Example: *healthy* patient classified as having a *disease*.

Imbalance: where accuracy fails



Let's suppose we have a dataset of **credit card transactions**.

We want to determine if a specific transaction is related to a **fraud** ($0 \rightarrow$ legitimate transaction, $1 \rightarrow$ fraud).

Fraud transactions are usually very rare, so our dataset might have a large imbalance (e.g., 99% of transactions in class 0, 1% of transactions in class 1=frauds).

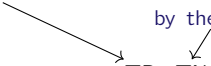
Q: *Can you think of a very straightforward classifier that achieves 99% accuracy?* **A:** $f(x) = 0$.

Accuracy in binary classification - another perspective

We can further develop accuracy like this:

TP = True Positives: items
of class 0 correctly classified
by the model

TN = True Negative: items
of class 0 correctly classified
by the model


$$Acc(y, \hat{y}) = \frac{TP+TN}{TP+FP+TN+FN}$$

This point of view enables us to further specify accuracy as equally considering the two types of errors/correct classification.

Balanced Accuracy

Balanced Accuracy

Instead of calculating accuracy on *all* of the samples, we calculate it separately for each category:

$$Acc^{(c)}(y, \hat{y}) = \frac{\sum_{i=1}^{n_c} 1[y_i = \hat{y}_i]}{n_c} = \frac{\sum_{i=1}^n 1[y_i = c] 1[y_i = \hat{y}_i]}{\sum_{i=1}^n 1[y_i = c]}$$

Data points in class c

The balanced accuracy is the average of the per-class scores:

$$BAcc(y, \hat{y}) = C^{-1} \sum_c Acc^{(c)}(y, \hat{y}) \quad (3)$$

For binary classification, we can rephrase it:

$$BAcc(y, \hat{y}) = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) = \frac{TPR + TNR}{2} \quad (4)$$

Other metrics for binary classification

Confusion Matrix

The confusion matrix is not a metric per se, but a collection of multiple metrics represented in tabular form:

		Ground truth	
		1	0
Predictions	1	True Positive	False Positive
	0	False Negative	True Negative

These elements have varying importance depending on the application, for example, medical systems usually require a low number of false positives. Other metrics are derived:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (5)$$

$$\text{Recall, Sensitivity, or TPR} = \frac{TP}{TP + FN} \quad (6)$$

Precision and Recall

Precision is the ratio between true positives and predicted positives ($TP + FP$).

It concentrates on **false positives**; it cannot measure false negatives.

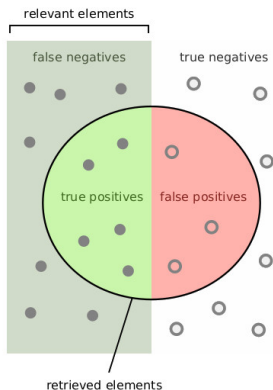
Recall or Sensitivity is the ratio between true positives and ground truth positives ($TP + FN$).

It concentrates on **false negatives**; it cannot measure false positives.

A high accuracy may be accompanied by low Precision or Recall, possible indication of imbalancedness in the dataset.

Specifically, high imbalance toward negative examples can be connected to high Accuracy and Precision, and very low Recall.

Precision and Recall



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Figure: Precision and Recall, modification of an image provided by MWarren us on Wikimedia Commons under license CC-SA.

A more complete picture for binary classification metrics

Sources: [26][27][28][29][30][31][32][33][34] view • talk • edit

		Predicted condition			
Actual condition	Total population = P + N	Positive (PP)	Negative (PN)	Informedness, bookmaker informedness (BM) = TPR + TNR - 1	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection	False positive rate (FPR), probability of false alarm, fail-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
Prevalence $= \frac{P}{P + N}$		Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR-) $= \frac{FNR}{TNR}$
Accuracy (ACC) $= \frac{TP + TN}{P + N}$		False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{PN}$ $= 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$		F ₁ score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes-Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \frac{\sqrt{TPR \times TNR \times PPV \times NPV}}{\sqrt{FNR \times FPR \times FOR \times FDR}}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

Table: Table from Wikipedia.

F-Score – Mediating between Precision and Recall

F-Scores mediate between Precision (P) and Recall (R):

F_1 -Score

F_1 -Score is the harmonic mean between P and R:

$$F_1 = 2 \frac{P \cdot R}{P + R} = \frac{2}{1/P + 1/R} \quad (7)$$

F_β -Score

F_β -score gives score β to R w.r.t. P:

$$F_\beta = (1 + \beta^2) \frac{P \cdot R}{\beta^2 P + R} \quad (8)$$

E.g., F_2 gives more (relative) importance to FN than FP; $F_{0.5}$ does the opposite.

Probabilistic binary classifiers – threshold

Most binary classifiers (we'll call them **probabilistic**) do not output directly the category, but output one value which represents a level of *confidence* or *probability* that a data point belongs to the positive category:

$$f(x) \in [0, 1], \quad f(x) = P(\underbrace{\hat{y}}_{\text{Predicted category}} = 1 \mid x)$$

The operation for assigning x to one of the two classes is by *thresholding* $f(x)$ using a value $\delta \in (0, 1)$:

$$\hat{y} = \begin{cases} 1 & \text{if } f(x) > \delta \\ 0 & \text{if } f(x) \leq \delta \end{cases}$$

By default, $\delta = 0.5$.

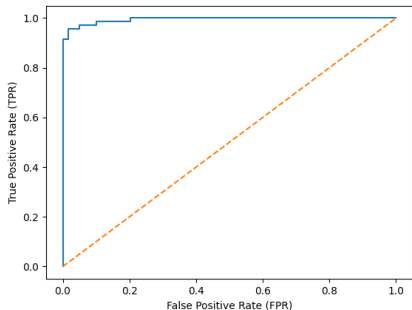
Receiver-Operating Curve (ROC)

If I vary δ from 0 to 1, I get different values for Precision, Recall (True Positive Rate—TPR), False Positive Rate (FPR), *etc.*

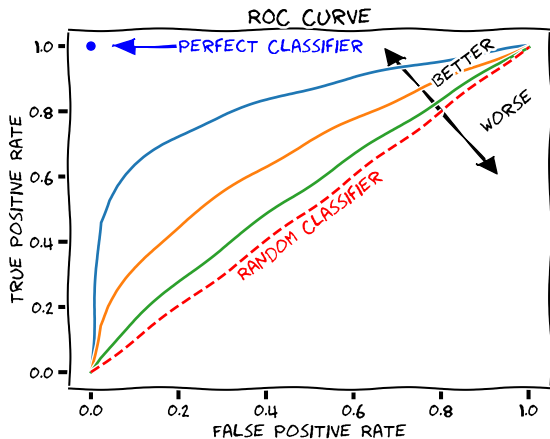
I can hence obtain a vector of pairs (TPR_k, FPR_k) for each value k of δ .

I can plot these value on a

chart with FPR on x and TPR on y.

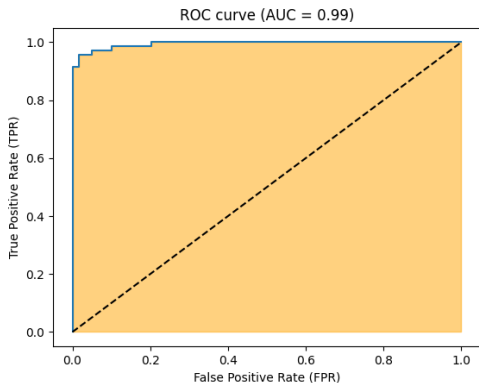


Receiver-Operating Curve (ROC)

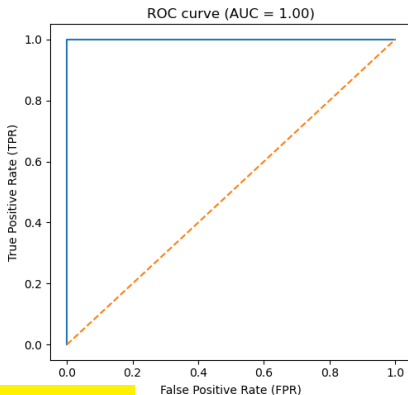
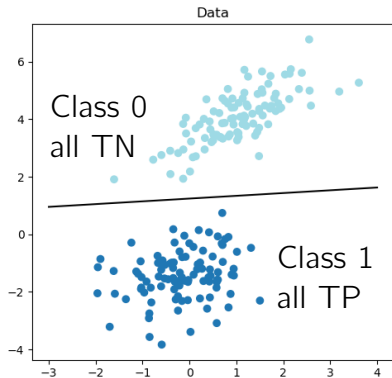


Area Under the [ROC] Curve (AUC or AUROC)

A common way to summarize ROC is by computing the Area Under the ROC Curve (AUC or AUROC):

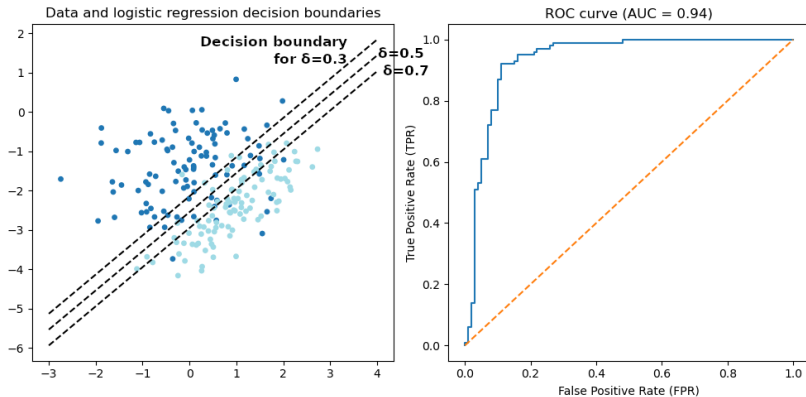


Inspecting ROC - Linear separability

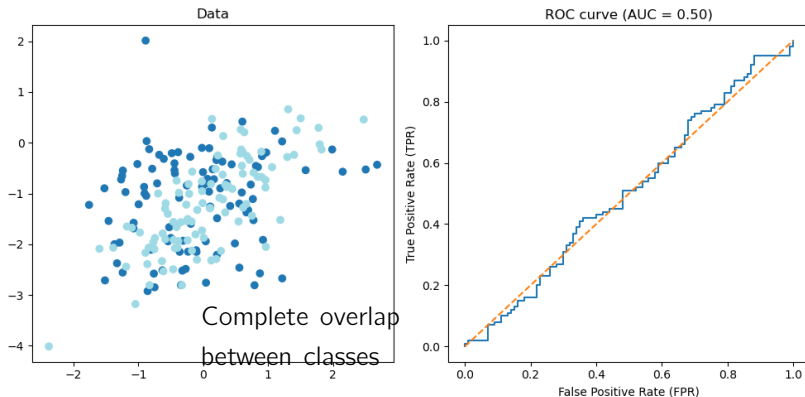


FPR=0, TPR=1 for each δ

Inspecting ROC - Imperfect linear separability



Inspecting ROC - Random classifier



Equal probability of committing a TPR or a FPR

Metrics for binary classification – summary

- In binary classification, we distinguish between errors: FP and FN (TP and TN are their “counterparts”).
 - FP and FN usually have different levels of importance depending on the specific problem.
 - In imbalanced problems, accuracy can be misleading.
- Accuracy is a metric which regards all errors—hence all classes—with the same importance.
- Precision, Recall (TPR), FPR, *etc.* all consider one type of error.
 - F_β -Score is a weighted average of these metrics, β can be toggled to give more importance to one or the other error type.
- The confusion matrix is a good way to *summarize* the performance on the two distinct types of errors.

Metrics for binary classification – summary

- Probabilistic binary classifiers (like linear regression) output a *probability* of assignment to positive class
- The class assignment happens by *thresholding* the probability (default = 0.5).
 - The threshold can be toggled to *vary* the classifications.
 - Accuracy, FPR, TPR, F-Scores, *etc.* vary as a consequence.
 - We can *measure* this variation in the ROC-curve (TPR vs. FPR).
 - ROC gives us an indication on the *overall* capability of the classifier in *telling apart* the two classes *regardless of the threshold used*.
 - It can be summarized in a scalar $\in [0, 1]$ by computing the Area Under the ROC Curve.

Metrics for binary classification – usage

- Accuracy is a good metric if:
 - You don't have an imbalanced dataset
 - You make no distinction between class 0 and 1.
- Precision, Recall, other similar metrics, and Confusion Matrices are easily interpretable ways to showcase the performance of the model on the different classes.
- F-Scores are also easily interpretable and are very good substitutes for accuracy (e.g., if one class matters more than the other)
 - They are also good metrics for determining the classification thresholds
- ROC curves can be a good way to summarize the overall performance of a classifier, but are hardly humanly-interpretable

Metrics for binary classification – usage

As a **generic suggestion**, when dealing with binary classification you should **always**:

- report **more than one metric** (i.e., don't stick to accuracy only)
- motivate **why** you prefer one or another (especially when using one of them for model selection or threshold selection)

Extension to $C > 2$

The output probabilistic classifiers in multi-class classification problems is different than the one of their binary counterparts.

For instance, for a four-way classification problem:

$$f(x) = \begin{pmatrix} 0.2 \\ 0.8 \\ 0.07 \\ 0.03 \end{pmatrix}$$

We link the prediction to the class in the following way:

$$\hat{y} = \arg \max_{j \in \{1, \dots, C\}} (f(x))$$

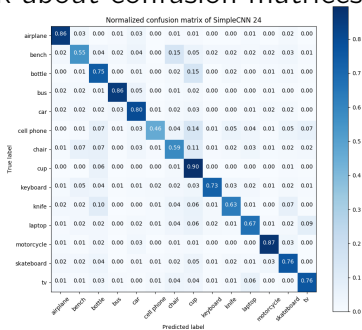
I.e., we pick the **index** of the highest element of $f(x)$.

Metrics for Multi-class classification

Accuracy and Balanced Accuracy are applicable in the same way.

There is no “positive” and “negative” class anymore, hence Precision, Recall, True Negative Rate, *etc.* don't make sense here.

We can still talk about confusion matrices:



Extending metrics for binary classification

We can still use metrics for binary classification in a per-class setting.

For each class we have a *one-vs-all* binary subproblem, for which we can compute the binary metrics:

Class	Precision
0	$P^{(0)} = \frac{\text{Correctly classified class 0}}{\text{All observations in class 0}}$
1	$P^{(1)} = \frac{\text{Correctly classified class 1}}{\text{All observations in class 1}}$
2	$P^{(2)} = \frac{\text{Correctly classified class 1}}{\text{All observations in class 1}}$

A similar extension can be done for recall.

Multi-label classification

In multi-label classification, we are not interested in classifying a datapoint in a *single* category, but a datapoint can belong in *multiple* categories at the same time.



$$f(x) = \begin{pmatrix} 0.7 \\ 0.4 \\ 0.8 \\ 0.1 \end{pmatrix} \begin{matrix} \text{cat} \\ \text{dog} \\ \text{bird} \\ \text{airplane} \end{matrix}$$

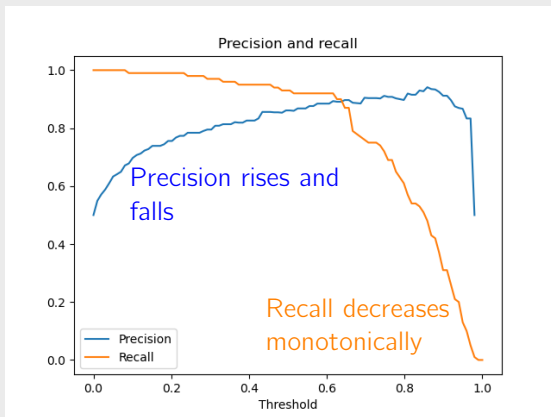
For each output category c , we can find a corresponding threshold δ_c such that

$$\hat{y} = \begin{pmatrix} \mathbf{1}[f_1(x) > \delta_1] \\ \vdots \\ \mathbf{1}[f_c(x) > \delta_c] \end{pmatrix}$$

Metrics – Multi-label classification

Precision and Recall

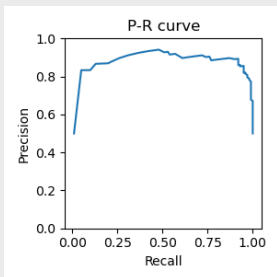
As done with the binary setting, we can toggle each δ_c separately and study its effect on metrics such as per-class Precision and Recall:



Metrics – Multi-label classification

P-R curve & Average Precision (AP)

We can turn this chart by recording the value of P for each value of R and plotting it. Remember: we do it separately for each class.



We can compute the area under the curve—analogously to what done with the ROC curve—and obtain another metric, AP. We have:

$$\begin{aligned} AP^{(c)} &= \text{Area under PR curve for class } c \\ &= \int_{r=0}^1 P^{(c)} @ [R^{(c)} = r] \end{aligned}$$

We can average through classes, obtaining the *mean Average Precision* (mAP):

$$mAP = C^{-1} \sum_c AP^{(c)}$$

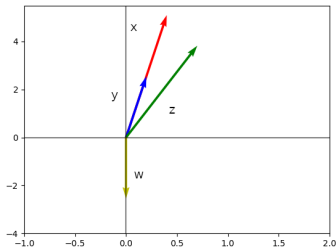
Loss Functions - Others

Cosine Similarity

Used to compare vectors.

$$L(y, \hat{y}) = \sum_i \frac{\text{dot}(\hat{y}_i, y_i)}{||\hat{y}_i|| ||y_i||}$$

Measures alignment between two vectors, without comparing magnitude (is the cosine of the angle between vectors):



$$L(x, y) = 1, \text{ total alignment}$$

$$L(x, z) = L(y, z) \approx 0.85$$

$$L(x, w) = L(y, w) \approx -0.45$$

What you must learn

- It is important to know the difference between metric and loss function.
- Metrics for regression: (R)MSE, MAE, R^2 , correlation coefficient. (R)MSE and MAE can also be used as loss functions. There exist other variants for outlier robustness.
- Metrics for binary classification: Accuracy, with all its cons. Confusion matrix and the two types of errors: what to they mean? All the metrics derived from the confusion matrix: Precision, Recall, F_β -scores, True Negative Rate, *etc.*
- The concept of probabilistic classifiers and how we can tweak the threshold δ to obtain different value for the metrics. The metrics derived from this tweaking: (AU)ROC, Precision-Recall curve and Average Precision.
- Extending these metrics to the multi-class setting. There exists also a multi-label setting.

Questions to Think About - Evaluation

1. What is the main feature that tells apart a loss function from a metric?
2. What happens if I use the difference between the predicted and true values (i.e., $\sum_i (y_i - \hat{y}_i)/n$) as a loss function?
3. Can accuracy be used to evaluate a regression model?
4. Why is accuracy a worse metric for binary classification than F_1 ?
5. Is precision more important than recall?

Take Home Messages - Evaluation

- Regression and classification metrics measure different properties of the model with respect to the data. Make sure you know what type of problem you are solving before choosing a metric.
- Metrics tell different facets of the same model.
 - It is useful to compute more than one metric in order to evaluate your choice.
 - Some metrics hint at some information about other metrics.
- Some metrics are easier to interpret than others. Choose wisely which to report (e.g., MSE vs. MAE, F_1 vs. AUROC) depending on the recipients of your report.
- Monitor and report both train and test metrics to spot possible over-/under-fitting.

Outline

① Cross Validation Methods

Overfitting

Cross Validation

② Evaluation and Metrics

This Section on CV

- This lecture covers the basics of cross-validation and how it relates to detecting overfitting.
- You will learn the basic concept of overfitting, how to detect it, and best practices for using cross-validation to detect overfitting.
- Additionally we cover Leakage and its importance in cross-validation.

Outline

① Cross Validation Methods

Overfitting

Cross Validation

② Evaluation and Metrics

Overfitting

- Overfitting is when the trained model memorizes undesirable patterns from training data.
- Like models learning noise in the data or irrelevant features.
- It reduces generalization performance. Models perform badly on the test set or on different data.
- Happens when model has too much learning capacity or it is trained for too long.

Overfitting

General Concept

Overfitting is when a model fails to generalize (make correct predictions).

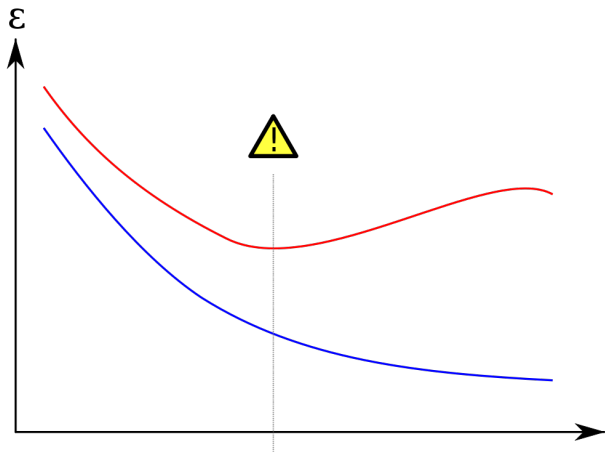
To detect it, look at the gap between train and validation/test losses.

$$L_{\text{gap}} = L_{\text{val}} - L_{\text{train}} \quad (9)$$

Overfitting is a continuous concept, not binary (yes or no overfitting). There are multiple "degrees" of overfitting.

An important concept is that all models overfit to some degree, but some are useful. Any model can also overfit, there is no model that completely prevents overfitting.

Overfitting



Blue is training loss, Red is validation/test loss.

Underfitting

Another phenomena related to overfitting, is underfitting.

This is when the training loss is high, and does not decrease during training. It can also happen that the training loss decreases a little but not enough for convergence (close to zero).

It basically means that the model does not fit the data. Could be model misspecification or incorrect model.

Outline

① Cross Validation Methods

Overfitting

Cross Validation

② Evaluation and Metrics

Model Assessment and Selection

- **Generalization.**
 - Prediction capability on independent test data.
 - How large is our prediction error?
- How to assess this performance?
 - Choice of hypothesis/model (linear vs non-linear) and choice of cost function/learning method depends on this!
- **Bias-Variance Trade-off.**
 - Balance between the accuracy of the estimation and its variance.

Components of Machine Learning algorithms

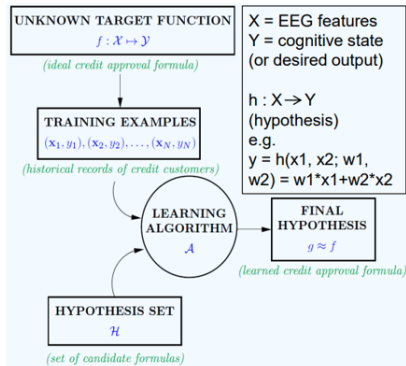


Figure by Y. S. Abu-Mostafa

Components of ML algorithms (learners):

Representation

- How knowledge is represented (formulas, programs, rules...)

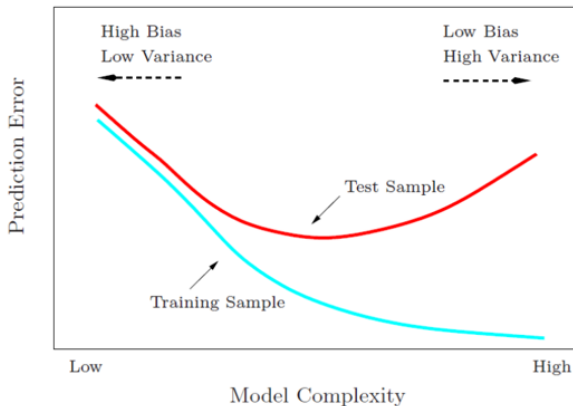
Evaluation

- Test of the learner (accuracy, cost/utility, ...)

Optimization

- Algorithm that finds the best representable hypothesis, according to the evaluation criterion

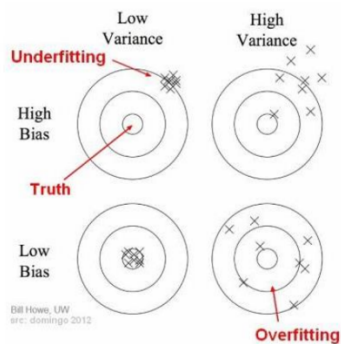
Model complexity



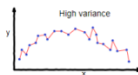
Tradeoff in complexity = tradeoff between bias and variance.

An algorithm can't be more complex and less complex at the same time!!

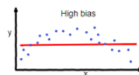
Under- and overfitting



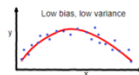
If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.



overfitting



underfitting



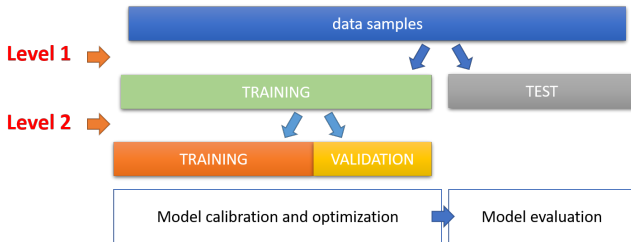
Good balance

How to detect and address overfitting?

- If our model does much better on the training set (0.01 loss) than on the test set (10.0 loss), then we are likely overfitting.
- Start with a very simple model to serve as a benchmark.
- Cross-validation.
- Training with more data (does not always work, unless the data is clean and relevant).
- Remove redundant features (dimensionality reduction).
- Regularization (penalty on the cost function to make the model simpler).

Model Selection: How to?

- Split data in test & training set



Cross Validation

Cross validation is the concept of evaluating a model on an independent dataset in order to evaluate its performance and assess the level of generalization.

This is usually done by using two datasets:

Training Set

The dataset where the model is trained and the training optimization process runs.

Metrics computed in this dataset are called training metrics.

Test Set

An *independent* evaluation dataset, ideally containing samples that are not present in the training set. Metrics computed in this dataset are called testing metrics.

Validation Datasets

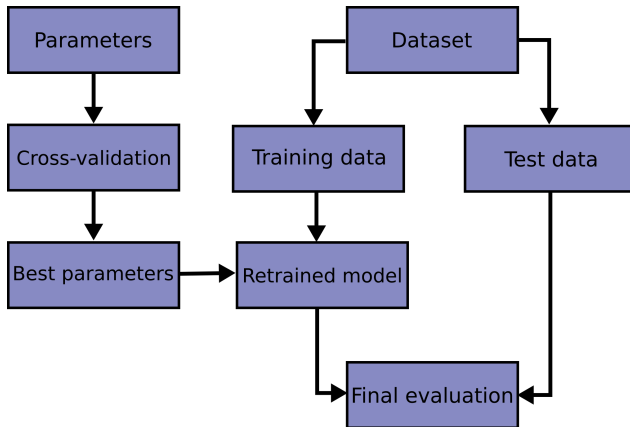
Additionally to a training and testing set, some methods require the use of a validation set.

When decisions need to be made during the training process, for example, hyper-parameter selection. Then all decisions *must* be made on the validation set.

Then the final evaluation metrics can be computed on the test set.

The training, validation, and testing sets, should all be independent to each other, and contain no samples in common.

Train/Validation/Test Workflow



Train/Validation/Test Workflow

The validation set is only needed when a decision needs to be made during the training and/or model development process.

After this process is finished, then final model can be trained on the combined training and validation set, this is usually called trainval or train-val dataset.

This allows to take advantage of the additional data of the validation dataset for training, but as with the test set, this step can be done only once. The model is trained on the train-val dataset, and evaluated in the test set, and final metrics presented on the test set.

Train/Test Splits

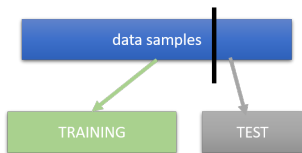
The big question is how to obtain training, validation, and test sets.

In the ideal case, these datasets would be captured and/or developed independently. But unfortunately this is rare, and typically one data source is used.

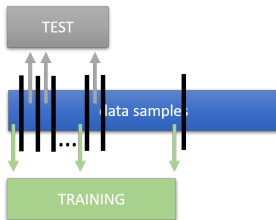
Then from a single dataset or data source, *random splits* can be made, so samples are selected to be part of the train, validation, or test set. This assumes that there are no duplicated samples in the dataset.

Types of data splits

- Chronological sampling



- Random sampling



Typical Train/Val/Test Ratios

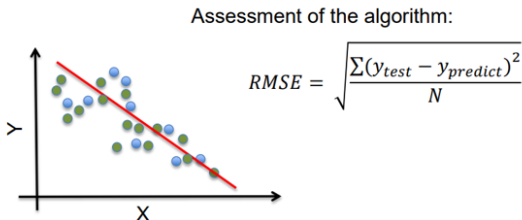
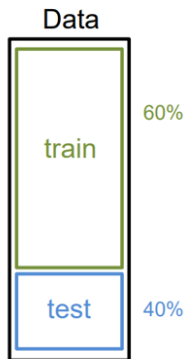
Here are some typical split ratios that are used in practice and in the literature.

- Training 80 %, Testing 20%.
- Training 80 %, Validation 10%, Testing 10%.
- Training 70 %, Validation 15%, Testing 15%.
- Training 60 %, Validation 20%, Testing 20%.

It is usual that validation and testing sets are the same size, and that the training set is the largest set.

Assessment one-cut

- Typical algorithm assessment (one cut)



The drawback of this one cut (60%-40%) approach is the insufficient data used for the assessment of the algorithm (only 40% is used for testing). So, cross-validation comes as a solution to increase the amount of testing data (artificially)!!

Assessment one-cut

- Typical algorithm assessment (one cut)



Data split differently



Assessment of the algorithm:

$$RMSE = \sqrt{\frac{\sum (y_{test} - y_{predict})^2}{N}}$$

The RMSE will most likely be different between the two types of splitting (evaluated on different test sets).

Leakage

Leakage is when samples from the training set *leak* and are part of the validation or testing datasets. This is undesirable because when evaluating your model, this will produce false or incorrect evaluation metrics. The model will look better than it is in reality.

Leakage can also happen when information related to the labels or not present at inference time, is presented during training. For example if somehow one of your features is the label that should be predicted.

Leakage is difficult to detect and could be very subtle. Only proper scientific practice can prevent this problem.

Preventing Leakage

User Data

When using people to generate data, assign different persons to train and test sets.

Time Series

This kind of data needs special care, as it is required to assign whole time series to either the train or test split, and not to divide a time series between both datasets.

Preventing Leakage

Objects

If your method is object-independent, then source objects should not be split across the train and testing sets. If you use objects A, B, and C, then A and C could be in the training set, and B in the test set.

Data Augmentation or Oversampling

This should only be applied on the training set, and only *after* any train/validation/test splits are made.

K-Fold Cross Validation

An issue with train/test splits is that these datasets are fixed. How does performance and generalization vary with varying data, or varying splits?

In K-Fold Cross Validation, a dataset is split into k folds (mostly equally sized). Then the following algorithm is used.

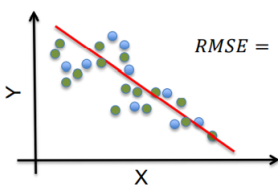
This means that K-Fold CV, you use each fold once for testing, and all folds minus the test one participate in training the model. This requires to train k models.

K-Fold Cross-Validation

Training data
split in iteration 1



Assessment of the algorithm:



$$RMSE = \sqrt{\frac{\sum (y_{validation} - y_{predict})^2}{N}}$$

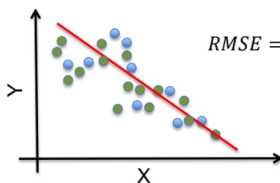
We slice the training data in (5) different folds, and train in one iteration on 4 folds (80%) and validate on the remaining 20% of the data.

K-Fold Cross-Validation

Training data
split in iteration 1



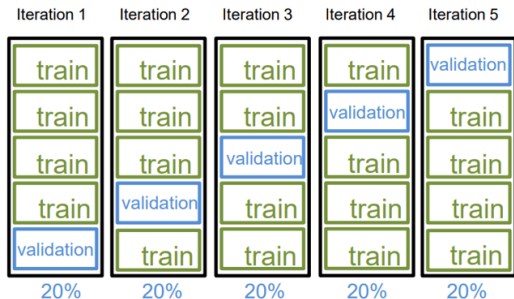
Assessment of the algorithm:



$$RMSE = \sqrt{\frac{\sum (y_{validation} - y_{predict})^2}{N}}$$

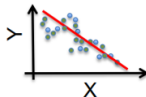
We slice the training data in (5) different folds, and train in one iteration on 4 folds (80%) and validate on the remaining 20% of the data.

K-Fold Cross-Validation



Assessment of the algorithm:

$$RMSE = \sqrt{\frac{\sum (y_{validation} - y_{predict})^2}{N}}$$

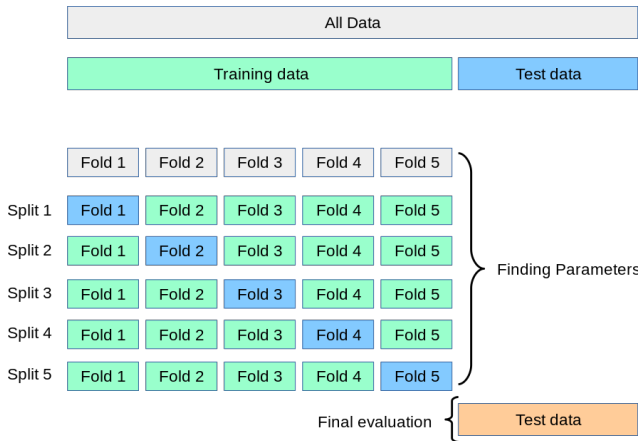


Using cross-validation we can assess our algorithm on the entire training data, as the average RMSE of all 5 iterations. After optimizing the hyperparameters on the validation sets, the model is retrained with all the training data and tested on the hold-out test set.

K-Fold Cross Validation

1. Results \leftarrow Array(k).
2. For $i \in [1, \dots, k]$:
 - 2.1 Train a model from scratch on folds $[1, \dots, k] - i$.
 - 2.2 Test on fold i and store result on Results[i]
3. Compute result mean as final cross-validation score.
4. Additionally the standard deviation of results can be used.

K-Fold Cross Validation



Leave One Out CV

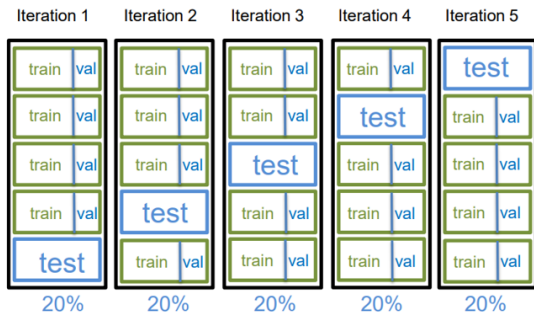
If we use $K = N$, then we produce a new method called Leave One Out Cross-Validation (LOO CV).

In this case, N models are trained, and each corresponding test set is only one sample.

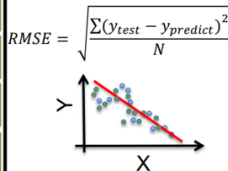
This method evaluates if the model is able to correctly predict all samples in the dataset.

Note that all Cross-Validation methods are means to evaluate a model, not to train a model for future use.

Nested Cross-Validation



Assessment of the algorithm:

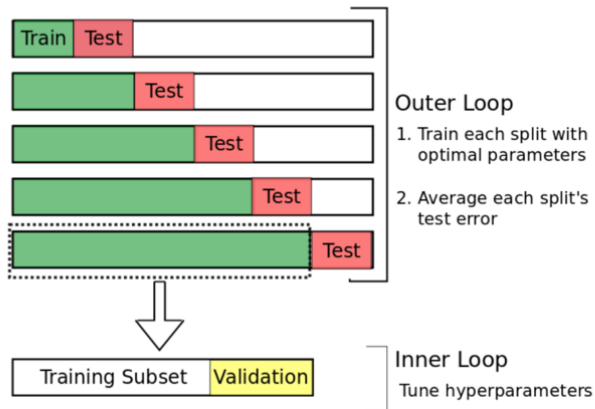


Using nested cross-validation we can assess our algorithm on the entire data, as the average RMSE of all 5 iterations. After optimizing the hyperparameters on the validation sets, the model is retrained in each iteration with all the training data and tested on the test set.

Time dependency

- Predicting the future with information from the past

Nested (chronological) cross validation



What you must learn

- The concept of Overfitting and Underfitting (also in the Statistical Learning Theory Lecture).
- Model Selection, Validation Datasets, Train/Test Splits.
- The concept of Leakage and when it can occur.
- The concept and methods for Cross-Validation.

Questions to Think About - Cross Validation

1. What are the differences between cross validation, k-fold cross-validation, and leave one out cross-validation?
2. What is the major difference between a validation and test sets?
3. Give an example (not ones from the lecture) of leakage between train and test set.
4. What is overfitting?
5. How can leakage be detected?
6. What should you considering when selecting cross-validation methods for your problem?

Take Home Messages - Cross Validation

- Proper training/validation/testing methodology is **extremely important** to obtain realistic measures of model performance.
- Leakage is a major problem that can happen without noticing. Papers are rejected due to this.
- Hyper-parameter tuning is also a major headache, as there is a trade-off between compute budget and finding the "right" hyper-parameters for your problem.

Questions?