TEST FOR DEV: Introduction to Data Science

WS 23/24, RWTH Aachen

February 5, 2024

1 Evaluation of Supervised Learning Problems

1.1 Need for evaluation and basic definitions

So far, we looked at the following supervised learning setting:

$$\text{descriptive features} \quad \begin{array}{c} \longrightarrow \\ \text{Naive Bayes Classifier} \\ \text{(Logistic) Regression} \\ \text{Support Vector Machines} \\ \text{Neural Networks} \\ \dots \\ \longrightarrow \end{array} \quad \text{target feature}$$

- We only distinguished between **training instances** and **unseen instances**
- An important introduced challenge is the occurrence of **over- and underfitting**

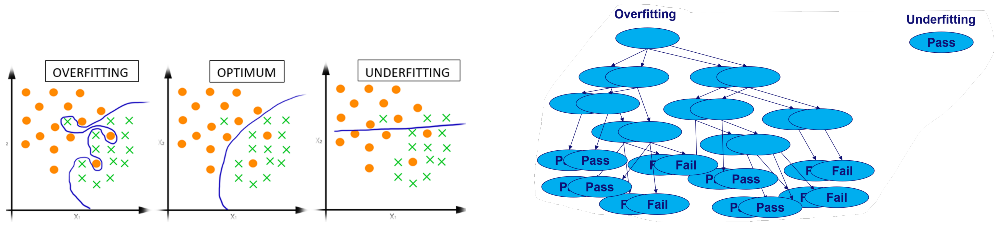


Figure 1.1: Over- and underfitting

Test set

What we now want to introduce is a **hold-out test set**. Instead of using all of the collected labeled training data for actual training, we split it into a "true" training set and a validation set, which is then used for parameter selection, hyperparameter tuning, stopping criteria, etc.

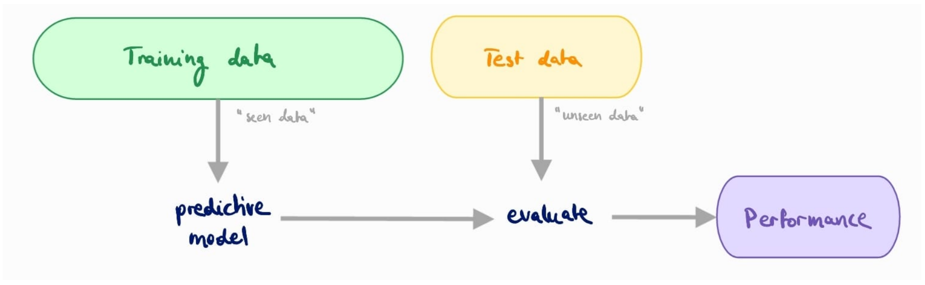


Figure 1.2: Idea of test set and performance evaluation

We're gonna look at an example of binary classification:

- After training the network, we can evaluate its performance using the **confusion matrix**
- For that, take instances of the test set, apply them on the network, and compare the calculated result and the target
- This leads to the possible combination of:

Confusion matrix

TP : True positive	result and target are both "positive"	FN : False negative	target is "positive", but the prediction "negative"
FP : False positive	target is "negative", but the prediction "positive"	TN : True negative	result and target are both "negative"

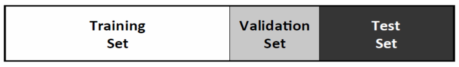
- For the confusion matrix, simply count the amount of TP, FN, FP, and TN occurrences over all test set instances
- We can then have the misclassification accuracy calculated as

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

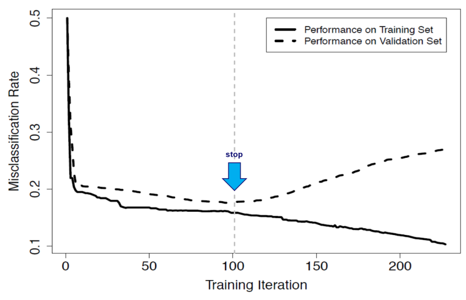
Validation set

Next, we'll look at the **validation set**. Instead of just having a test set, it is also possible to have a validation set that "pre-evaluates" the trained model. This again is for parameter optimization, model selection, or stopping criterion.

- One may train hundreds of models using training data, and then select one model that performs well on the validation set, or
- One can stop iterating before the model starts to overfit the training data ("self-reflection")



Typical splits are $50 : 20 : 30$ or $40 : 20 : 40$



Validation set is used to avoid overfitting the test set

Figure 1.3: Idea of test set and performance evaluation

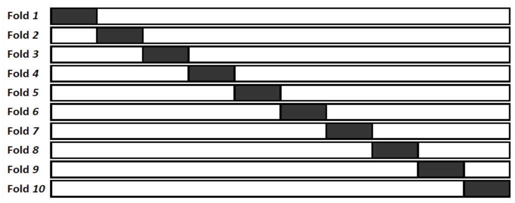
From now on, we will abstract from the validation set and only consider training and test data.

1.2 Cross validation

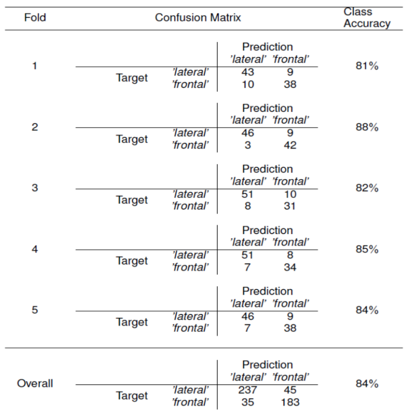
For k**-fold cross-validation**, we consider different parts of the dataset for testing, as can be seen in 1.4. If the accuracy stays stable, one knows that the model is probably good and doesn't depend on certain parts of the training set.

k-fold cross-validation

- Rule of thumb: typically 10 is a good number for k



Split of test set (dark parts) for folds, light parts are training data



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

Result of 5-fold cross validation

Figure 1.4: k-fold cross-validation

One special case of k-fold cross-validation is **leave-one-out cross-validation** where $k = 1$.

Leave-one-out cross-validation

- This is also known as **jackknifing**
- The test set is then only a single instance, and the approach is only used when less data is available

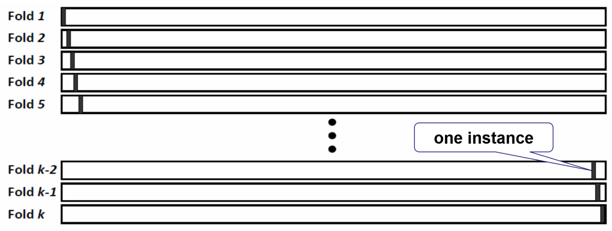


Figure 1.5: Leave-one-out cross-validation

Another cross-validation technique is **bootstrapping** where Repeatedly (k times), m random instances are selected as test set.

Bootstrapping

- This is typically used for smaller data sets, but k is usually much higher than for the k-fold technique

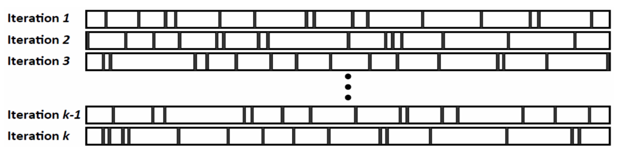


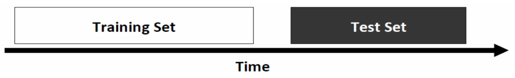
Figure 1.6: Bootstrapping

Next, we'll compare random and out-of-time sampling.

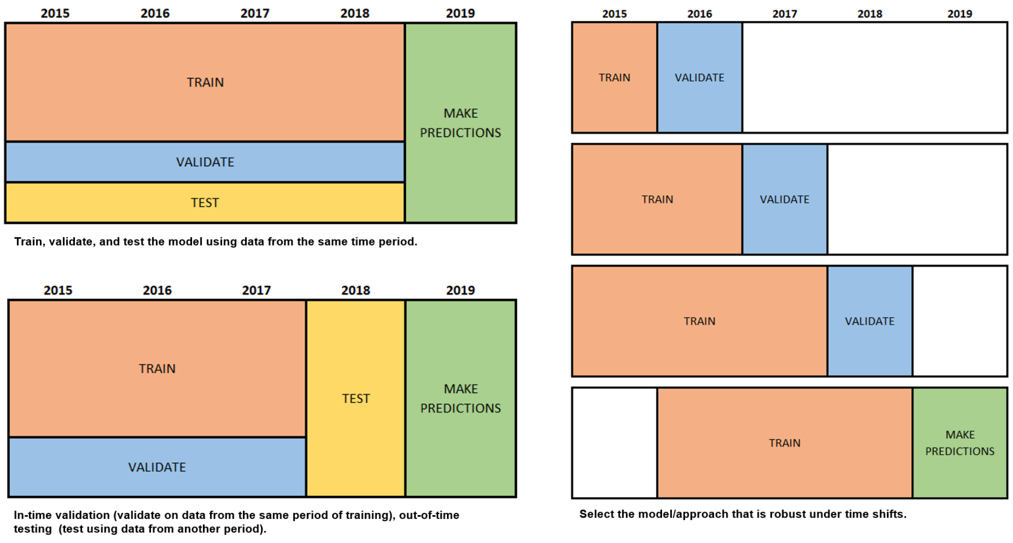
Random sampling

- For **randomm sampling** it goes that it works best if the instances are independent.
- But in the case of **context drift**, the time dimension plays a key role (imagine seasonal effects, backlog, ...)
- → random sampling gives misleading good results if processes drift
- → use data from an entirely different time period, which is called **out-of-time sampling**

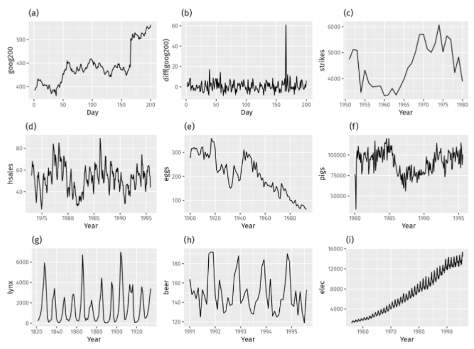
Out-of-time sampling



Idea of out-of-time sampling


2015
2016
2017
2018
2019
TRAIN
VALIDATE
TEST
MAKE PREDICTIONS
Train, validate, and test the model using data from the same time period.
In-time validation (validate on data from the same period of training), out-of-time testing (test using data from another period).
Select the model/approach that is robust under time shifts.

Different combinations (can train multiple models and select best one)



- Models may degrade over time due to drifts
- $\rightarrow$ need to decide if and when to re-train the model
- Tradeoff: having enough data $\leftrightarrow$ adapting to trends

Figure 1.7: Concept drift

1.3 Categorical target features

We're gonna start with a **boolean target**, for which we already mentioned the **confusion matrix** before, but gonna go into more detail now.

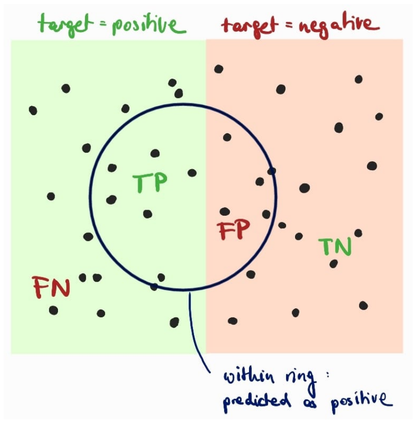


Figure 1.8: Confusion matrix visualized on dataset

With TP, FP, FN, and TN we cam make some further definitions:

- Type I error — FP is also known as **type I error**, or false alarms
- Type II error — FN is also known as **type II error**, or missed alarms
- Further we have some rates:

TPR — True positive rate : $TPR = \frac{TP}{TP+FN} = 1 - FNR$ — fraction of positives classified as positive

TNR — True negative rate : $TNR = \frac{TN}{TN+FP} = 1 - FPR$ — fraction of negatives classified as negative

FNR — False negative rate : $FNR = \frac{FN}{FN+TP}$ — fraction of positives classified as negative

FPR — False positive rate : $FPR = \frac{FP}{FP+TN}$ — fraction of negatives classified as positive

- Further, we define:

Accuracy — Accuracy : $ACC = \frac{TP+TN}{TP+TN+FP+FN}$ — Missclassifications : $1-ACC$

Precision — Precision : $precision = \frac{TP}{TP+FP}$ — within circle, fraction of TP

Recall — Recall : $recall = \frac{TP}{TP+FN} = TPR$ — green part, fraction of TP

F_1-measure — F_1-measure : $F_1 = 2\frac{prec \cdot recall}{prec+recall} = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$ — harmonic mean of precision and recall

Sensitivity — Sensitivity : $sensitivity = \frac{TP}{TP+FN} = TPR = recall$

Specificity — Specificity : $specificity = \frac{TN}{TN+FP} = TNR$ — red part, fraction of TN

With these measures, we can now evaluate the data. Consider e.g. **imbalanced data** where we have:

- 99% in a dataset of size 100 is positive ($\rightarrow$ 99 instances positive, 1 negative)

- Our classification always predicts a positive label
- We then have:

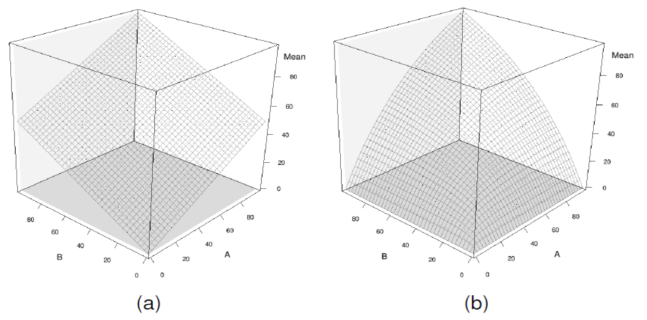
$$precision = \frac{99}{99+1} = 0.99 \qquad recall = \frac{99}{99+0} = 1$$

- When we now apply simple average class accuracy, where each class has the same weight independent of size, we get the following value:

$$\begin{aligned} average\ class\ ACC &= \frac{1}{|classes|} \sum_{c \in classes} recall_c \\ &= \frac{1}{2}\left(\frac{99}{99+0} + \frac{0}{0+1}\right) = 0.5 \end{aligned}$$

- When we instead use a **harmonic mean**, again with each class having the same weight independent of size, we get:

$$\begin{aligned} average\ class\ ACC_{HM} &= \frac{1}{\frac{1}{|classes|} \sum_{c \in classes} \frac{1}{recall_c}} \\ &= \left(\frac{1}{2}\left(\frac{99+0}{99} + \frac{0+1}{0}\right)\right)^{-1} = \text{"0.0"} \end{aligned}$$

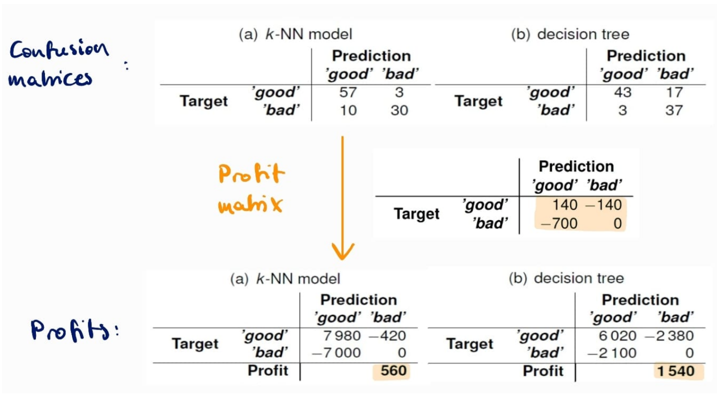


For HM (visualized in b), both A and B need to be good for the HM to be good

Figure 1.9: Arithmetic and harmonic mean

We can not only use the confusion matrix by itself but also weigh or quantify good and bad. This is then the **profit matrix**. Profit matrix

- We then have the entries TP_{profit}, FP_{profit}, FN_{profit}, and TN_{profit}
- By adding a profit quantification, we can express that FP and FN don't always have the same costs (e.g. car accident)


Confusion matrices :
Profit matrix
Profits:

(a) *k*-NN model

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

(b) decision tree

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

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

(a) *k*-NN model

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

(b) decision tree

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

Figure 1.10: Profit matrix principle

1.4 Receiver Operating Characteristic Curve

Before discussing the curve, we'll look at what we evaluate.

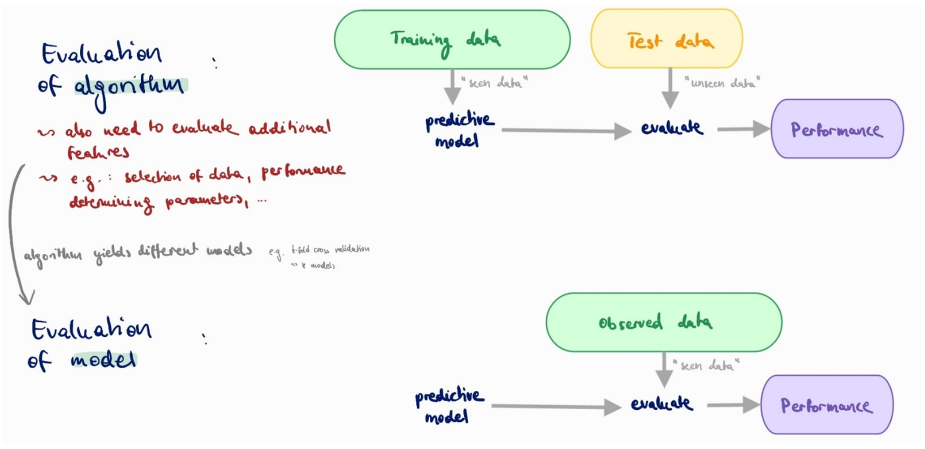

Evaluation of algorithm :
⇝ also need to evaluate additional features
⇝ e.g.: selection of data, performance determining parameters, ...
algorithm yields different models
e.g. k-fold cross validation
⇝ k models
Evaluation of model :
Training data
"seen data"
Test data
"unseen data"
predictive model
evaluate
Performance
Observed data
"seen data"
predictive model
evaluate
Performance

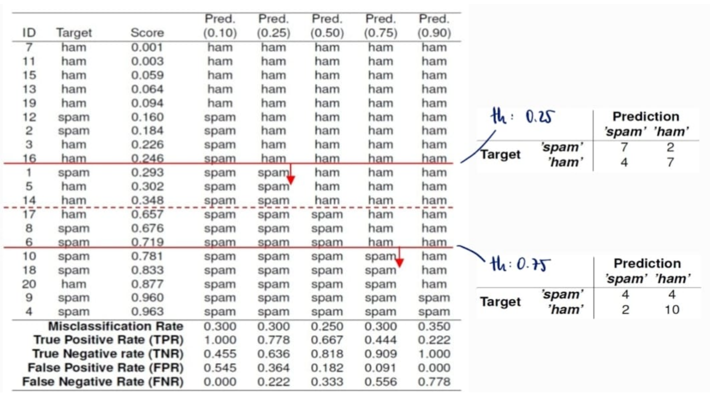
Figure 1.11: What is evaluated?

Now it could happen, that we only have positive observations (common e.g. in process discovery). This can originate from the ”survival bias” which describes that positive examples are more likely to be recorded. FP and TN can then of course not be calculated, and the evaluation is just as the training more difficult.

Prediction score

Now, let's move on the the ROCC. Assume we have a non-binary outcome with the resulting value being between 0.0 and 1.0, the **prediction score**.

- 0 means ”pretty sure” the predicted class is negative, 1 means ”pretty sure” the predicted class is positive
- The question for all in-between values (especially 0.5): which class are they assigned to, and where to put the threshold?
- By playing with the threshold, different confusion matrices are created.



ID	Target	Score	Pred. (0.10)	Pred. (0.25)	Pred. (0.50)	Pred. (0.75)	Pred. (0.90)
7	ham	0.001	ham	ham	ham	ham	ham
11	ham	0.003	ham	ham	ham	ham	ham
15	ham	0.059	ham	ham	ham	ham	ham
13	ham	0.064	ham	ham	ham	ham	ham
19	ham	0.094	ham	ham	ham	ham	ham
12	spam	0.160	spam	ham	ham	ham	ham
2	spam	0.184	spam	ham	ham	ham	ham
3	ham	0.226	spam	ham	ham	ham	ham
16	ham	0.246	spam	ham	ham	ham	ham
1	spam	0.293	spam	spam	ham	ham	ham
5	ham	0.302	spam	spam	ham	ham	ham
14	ham	0.348	spam	spam	ham	ham	ham
17	ham	0.657	spam	spam	spam	ham	ham
8	spam	0.676	spam	spam	spam	ham	ham
6	spam	0.719	spam	spam	spam	ham	ham
10	spam	0.781	spam	spam	spam	spam	ham
18	spam	0.833	spam	spam	spam	spam	ham
20	ham	0.877	spam	spam	spam	spam	ham
9	spam	0.960	spam	spam	spam	spam	spam
4	spam	0.963	spam	spam	spam	spam	spam
Misclassification Rate			0.300	0.300	0.250	0.300	0.350
True Positive Rate (TPR)			1.000	0.778	0.667	0.444	0.222
True Negative rate (TNR)			0.455	0.636	0.818	0.909	1.000
False Positive Rate (FPR)			0.545	0.364	0.182	0.091	0.000
False Negative Rate (FNR)			0.000	0.222	0.333	0.556	0.778

th: 0.25

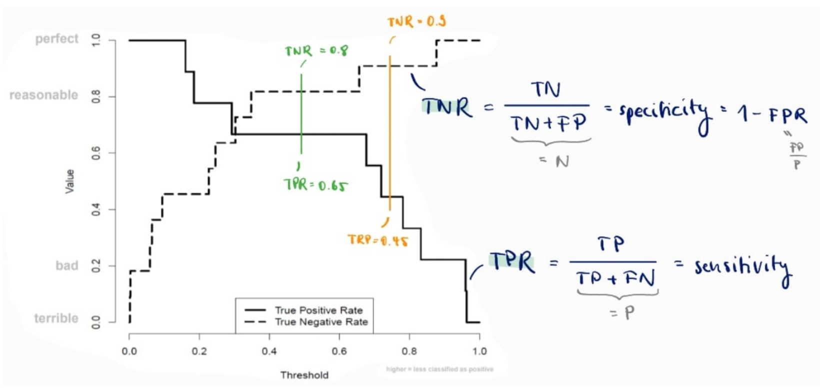
		Prediction 'spam'	'ham'
Target	'spam'	7	2
	'ham'	4	7

th: 0.75

		Prediction 'spam'	'ham'
Target	'spam'	4	4
	'ham'	2	10

Figure 1.12: Playing with prediction score threshold (and resulting confusion matrix)

To evaluate which threshold to choose, look at the TNR and TPR as in 1.13



Goal: both TNR and TPR should be as high as possible, but there is a clear trade-off

- Sensitivity (TPR) as the fraction of positives classified as positive
- Specificity (TNR) as the fraction of negatives classified as negative

Figure 1.13: How to choose prediction score threshold

Using these values, we can now draw a **receiver operating characteristic** curve. We have the goal to beat the random guessing line.

Receiver Operating Characteristic Curve

- Define q as the actually positive fraction, and $1 - q$ the actually negative fraction
- With probability p we now guess that an instance is positive, and with $1 - p$ that an instance is negative[1]

$$\implies TP = pq \qquad TN = (1-p)(1-q)$$
$$FP = p(1-q) \qquad FN = (1-p)q$$

$$\implies TPR = \frac{TP}{TP+FN} = \frac{pq}{pq + (1-p)q} = p$$

[1]Following terms are only fractions, multiply with N to get counts

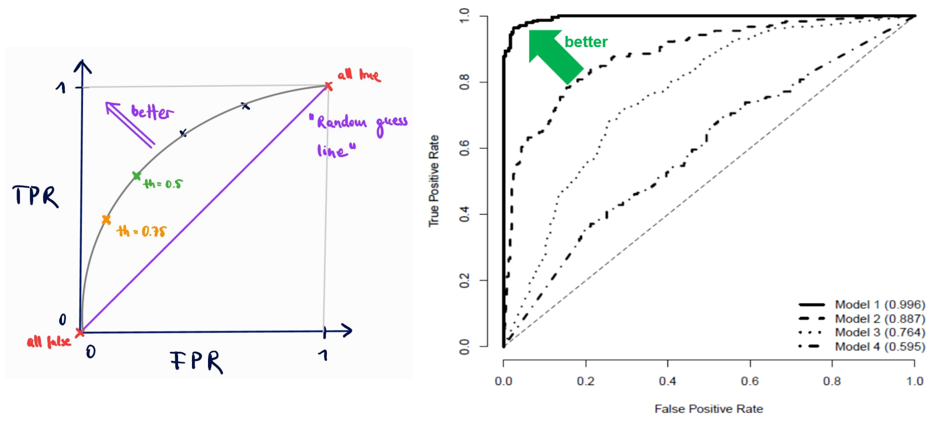


Figure 1.14: Receiver Operating Characteristic Curve

$$TPR = \frac{FP}{TN + FP} = \frac{p(1-q)}{(1-p)(1-q) + p(1-q)} = p$$

- This means: for random guessing as we described here, we have $TPR = p = FRP$ (where q doesn't matter)
- To be better than this random guess, we need to maximize the **area under the curve (AUC)**

Area Under The Curve

$$ROC\ index = \sum_{i=2}^{|T|} \underbrace{(FPR(T[i]) - FPR(T[i-1]))}_{\text{steps } x\text{-axis}} \cdot \underbrace{\tfrac{1}{2}(TPR(T[i]) - TPR(T[i-1]))}_{\text{average height } y\text{-axis}}$$

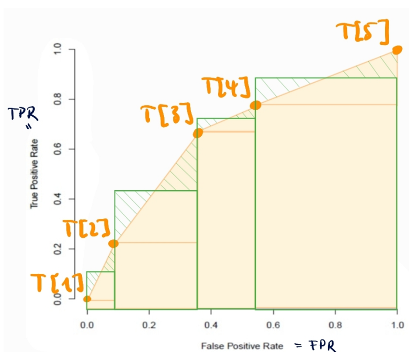
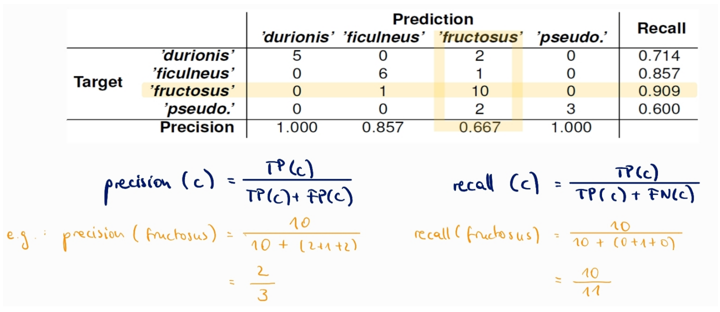


Figure 1.15: Area Under the Curve (AUC)

1.5 Multinomial targets

Now, we consider not only boolean but also multinomial categorical targets. As we already indicated before, we then have a precision and recall value per target class. Let's look at one exemplary confusion matrix.



		Prediction				Recall
		'durionis'	*'ficulneus'*	*'fructosus'*	*'pseudo.'*	
Target	*'durionis'*	5	0	2	0	0.714
	'ficulneus'	0	6	1	0	0.857
	'fructosus'	0	1	10	0	0.909
	'pseudo.'	0	0	2	3	0.600
	Precision	1.000	0.857	0.667	1.000	

$$\text{precision}(c) = \frac{TP(c)}{TP(c) + FP(c)} \qquad \text{recall}(c) = \frac{TP(c)}{TP(c) + FN(c)}$$

e.g.: $\text{precision}(\text{fructosus}) = \frac{10}{10 + (2+1+2)} = \frac{2}{3}$ $\qquad$ $\text{recall}(\text{fructosus}) = \frac{10}{10 + (0+1+0)} = \frac{10}{11}$

Figure 1.16: Exemplary confusion matrix for multinomial target

1.6 Continuous target features

Since we now can't count the "positive" vs "negative" predictions, we need to introduce another error measure. We have the following to our deposit given the actual target value t_i, the descriptive features x_i and the predicted value $\mathbb{M}(x_i)$:

$$\text{Sum of squared errors}: \frac{1}{2}\sum_{i=1}^{n}\big(t_i - \mathbb{M}(x_i)\big)^2$$

$$\text{Mean squared error}: \frac{1}{n}\sum_{i=1}^{n}\big(t_i - \mathbb{M}(x_i)\big)^2$$

$$\text{Root mean squared error}: \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(t_i - \mathbb{M}(x_i)\big)^2}$$

$$\text{Mean absolute error}: \frac{1}{n}\sum_{i=1}^{n}\text{abs}\big(t_i - \mathbb{M}(x_i)\big)$$

easy interpretation as average (absolute) error

From these known errors, we can derive a new idea: the R^2 **coefficient**: R^2

$$R^2 = 1 - \frac{\textit{sum of squared errors}}{\textit{total sum of squares}}$$

$$\text{with } \textit{total sum of squares} = \frac{1}{2}\sum_{i=1}^{n}\big(t_i - \bar{t}\big)^2, \text{ for average target } \bar{t}$$

- This coefficient compares the performance with guessing the overall average
- Value interpretation (range typically $[0, 1]$)

1 : perfect score, all predictions are perfect
0 : terrible score, not better than baseline (guessing average)
< 0 : even worse than guessing mean

1 (perfect score) to 0 (terrible score)

Just as before, we can use the following cross-validation techniques:

- k-fold cross-validation
- Leave-one-out cross-validation ($k = 1$)
- Bootstrapping (k times, leave out random m elements)

- Out-of-time sampling

So as we saw, to abstract from boolean classification to more complex targets, we do:

- Boolean confusion matrix measures for each class if the targets are multinomial
- A R^2-coefficient evaluation (or also an evaluation using previously introduced errors), which also shows how much better than average guessing we are for continuous output targets

1.7 A/B Testing

Recommendations

So far, we only looked at supervised learning. In this learning setting, we turn data into predictive models. Next, we want to **turn predictions into recommendations**:

- The predictions are about whether we fail or not. So, if you will fail, start working.
- Then, if you "change the descriptive features" (so the input), we predict a more desirable outcome.
- This concept can also be adapted to "customers like you ended up buying", etc.

Our main issues for recommendations are:

- What is cause and what is effect? What is controllable, and what isn't? (E.g.: wealthy people drive Porsches, so let's buy one; if I wear sunglasses, it will not rain)
  $\Longrightarrow$ correlation doesn't imply causation
- Recommendations change the reality they are based on

A/B testing

We often have hidden variables (e.g. explaining the following correlations: eating more ice cream leads to more criminal behavior, using an umbrella leads to higher walking speeds). To identify them, we introduce **A/B testing**. For that, we randomly offer two **variants**:

$$A \longleftrightarrow B$$

- For example: intervene based on prediction or not
- Then we conduct statistical hypothesis testing (decide whether data at hand sufficiently supports a particular hypothesis)

1.8 Concept drift

Concept drift

The problem of **concept drift** describes how a process changes over time. A good example is ice cream sales throughout one year. As one can see in 1.17: Within one year the curve heavily peaks over the summer months, but the curve is stable over the years.

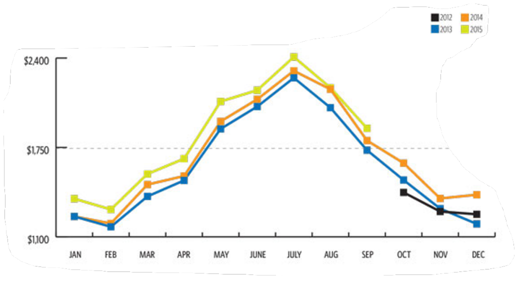
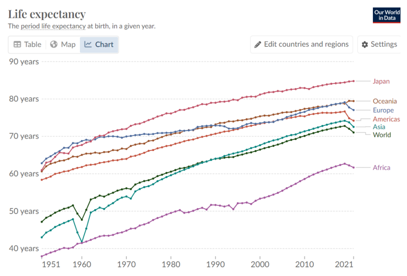


Figure 1.17: Process changing over time: example ice cream sales

Whenever trying to describe a model, the following questions are important:

- What should the model describe? Which time frame? (E.g. last year, last decade, ...)
- When predicting, is it better to use only recent data or also include older data? (drift $\leftrightarrow$ sparsity)



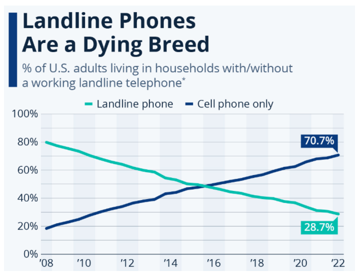


Figure 1.18: Concept drift: time dependency examples

To have a measure of how much drift is even involved in given data, we use the **system stability index (SSI)**.

$$SSI = \sum_{c \in classes} \left(\underbrace{\frac{|\mathcal{A}_{t=c}|}{|\mathcal{A}|}}_{\substack{\text{fraction of instances} \\ \text{in } \textbf{original test set} \\ \text{classified as } c}} - \underbrace{\frac{|\mathcal{B}_{t=c}|}{|\mathcal{B}|}}_{\substack{\text{fraction of instances} \\ \text{in } \textbf{new data set} \\ \text{classified as } c}} \right) \cdot \log_e \left(\frac{|\mathcal{A}_{t=c}|}{|\mathcal{A}|} \Big/ \frac{|\mathcal{B}_{t=c}|}{|\mathcal{B}|} \right)$$

System Stability Index

- Informal interpretation: $\begin{array}{rl} SSI < 0.1 & \text{no significant drift} \\ 0.1 \leq SSI < 0.25 & \text{moderate drift} \\ 0.25 \leq SSI & \text{significant drift} \end{array}$

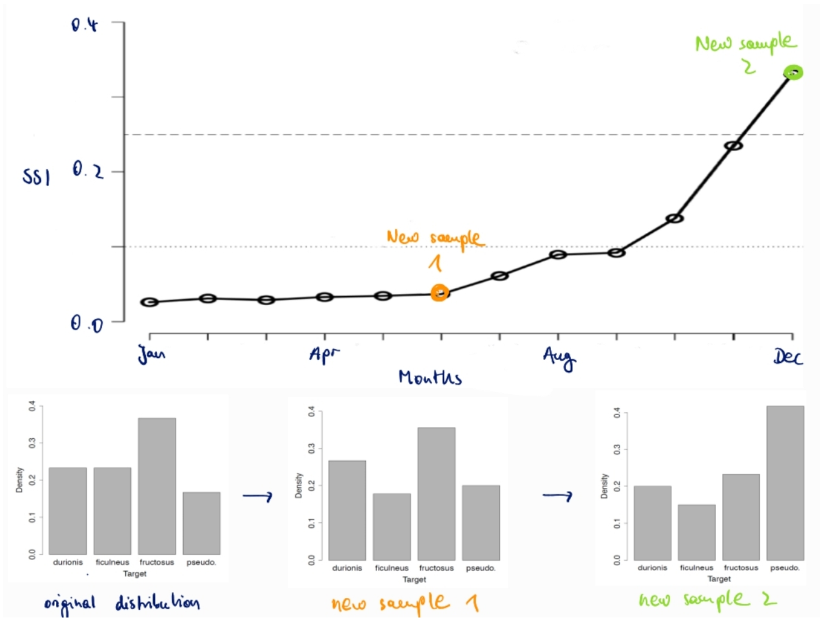
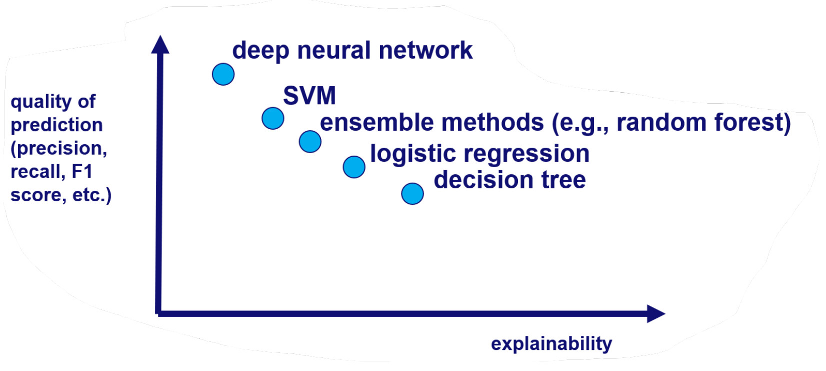


Figure 1.19: System Stability Index: example

1.9 Explainability

A final interesting point in comparing different SL-techniques is their explainability. Figure 1.20 shows a potentially existing trade-off, where more complex and therefore less explainable techniques have a better quality of prediction.


quality of prediction (precision, recall, F1 score, etc.)
deep neural network
SVM
ensemble methods (e.g., random forest)
logistic regression
decision tree
explainability

Positions are just indications, true positions depend on many factors (data set, parameters, ...)
DNN can perform very poorly on small data sets

Figure 1.20: Quality of prediction vs. explainability

As one can see: especially DNNs are not explainable. Whether the system derives:

- Guilty because of $X\&Y\&Z$, or
- If $X\&Y\&Z$, then guilty

Can't be told so far (so not really good judges $\rightarrow$ true reasoning not tellable).

Similar example: using an NN as a doctor can lead to it telling you that you have an 80% chance of dying, but not why IF it is poorly developed and applied.