

# Performance Evaluation

How to assess model quality in absolute numbers?

$$\hat{f} = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \operatorname{Emp}(f)$$

Risk after training  $\sum_{(x,y) \in \mathcal{D}_{\text{train}}} L(y, \hat{f}(x))$  which we minimize during optimization for models of hypothesis space  $\mathcal{H}$  can be overly optimistic, prone to overfitting when we choose model that just has the lowest training error

But "best" model should produce the most meaningful predictions on new unseen data  $\mathcal{D}_{\text{test}}$  (while we usually don't have access to fresh data)  
"out of sample testing", true future performance

$\Rightarrow$  Divide dataset manually into  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{test}}$ . Use  $\mathcal{D}_{\text{test}}$  to estimate Generalization Error, i.e. to calculate expected loss on new observation

Generalization Error of Fixed Model:  $GE(\hat{f}, L) = \mathbb{E}_{x,y} [L(y, \hat{f}(x))]$  assuming  $\mathcal{D}_{\text{test}}$  drawn i.i.d. from same distribution as  $\mathcal{D}_{\text{train}}$ ; independent of observations from  $\mathcal{D}_{\text{train}}$   
Expectation over a single random test point  $(x,y) \sim P_{X,Y}$

Estimator for GE using dedicated test set of size  $m$ :  $\hat{GE}(\hat{f}, L) = \frac{1}{m} \sum_{(x,y) \in \mathcal{D}_{\text{test}}} L(y, \hat{f}(x))$  type of Monte Carlo Integration

Rarely possible, in the end we do fit chosen model on all data (best result for  $\hat{f}$ ), so final model can't really be used in construction of practical GE-estimator  
 $\hookrightarrow$  Technically there is no data left to test!  $\Rightarrow$  instead we evaluate the next best thing, the Learner itself  
unseen

Sometimes we want to use different measure to evaluate model/learner performance than what we used to fit the model (Loss function, ERM)  
That's why we differentiate two types of losses. Computational advantages push us to use different Loss measures or simple implementation requirements.

**Inner Loss:** Loss function used for ERM  $\rightarrow$  model fit (pointwise)

**outer loss:** Loss function used for performance evaluation of inner loss fitted model (not necessarily point wise)

Inner loss must be differentiable w.r.t. parameters  $\Theta$ . Outer Loss may be non-differentiable or even something abstract like ratio from confusion table

Example: Logistic Regression uses binomial inner Loss and 0-1 loss outside e.g. number of misclassifications/ $n_{\text{test}}$   
then our also not convex  $\Rightarrow$  not an option for optimization  
when used inside for ERM it's a NP hard problem, discrete optimization

**Set-Based Performance Metrics** Prediction quality measure only defined on complete test set, not for single observations

Scalar perf. metric  $pe: \bigcup_{m \in \mathbb{N}} (Y^m \times \mathbb{R}^{m \times g}) \rightarrow \mathbb{R}, (y, \hat{f}) \mapsto pe(y, \hat{f})$   $m \triangleq$  sample size of test dataset  $|\mathcal{D}_{\text{test}}| = n_{\text{test}}$

Prediction matrix  $F$  is defined as  $F := \begin{bmatrix} \hat{f}(x^{(1)}) \\ \vdots \\ \hat{f}(x^{(m)}) \end{bmatrix} \in \mathbb{R}^{m \times g}$   
 $\hat{f}$  refers to number of scoring functions same for linear regression  
In binary classification it's actually just one, leading to scalar  $\hat{f}(x^{(i)})$   
In multiclass case  $\hat{f}(x^{(i)})$  denotes row vector with  $g$  entries  $\hat{g}$  score outputs

Pointwise Loss can be seen as special case  $pe_L(y, \hat{f}) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{f}(x^{(i)})) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \hat{f}(x^{(i)}))$   
just average loss over  $\mathcal{D}_{\text{test}}$

**Generalization Error of Inducer:** Let  $t_n$  be a Learner on  $n_{\text{train}}$  points from  $P_{X,Y}$ , then we evaluate perf. quality of models that are fitted with  $t_n$

$\Rightarrow GE(t_n, \gamma, n_{\text{train}}, pe) := \lim_{n_{\text{test}} \rightarrow \infty} \mathbb{E}[pe(y, \hat{f}_{\mathcal{D}_{\text{test}}, t_n(\mathcal{D}_{\text{train}}, \gamma))}]$   
Some  $pe$  might only converge for  $n_{\text{test}} \rightarrow \infty$  (set-based  $pe$ )  
using pointwise losses  $pe_L$  (no limit needed)  
yields  $\hat{f} \in \mathcal{H}$ , may write  $\hat{f}_{\mathcal{D}_{\text{train}}}$   
integrating out over  $x, y$  with  $\hat{f} = t_n(\mathcal{D}_{\text{train}}, \gamma)$   
Expected value over both  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{test}} \triangleq$  over all possible randomly sampled training & test data sets, averaged.  
Learner-GE captures randomness of full dataset in a certain way through stochasticity of Learning algo and test sample  
 $GE(\hat{f}, L) = GE(t_n, \gamma, n_{\text{train}}, pe_L(\cdot, \cdot))$

- Problem: This assumes independent random sampling of all  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{test}}$  from  $P_{X,Y}$ , lots of randomness associated if we use single holdout split  
But interestingly:  $n_{\text{train}} \uparrow$  tends to increase train error and decrease only test error (pattern learning easier with less data points - we should not base  $n_{\text{train}}$  on this effect!)
- More data is always better, increases conf. We want  $n_{\text{train}} \uparrow$  but for more certainty in Learner-GE estimate we also want larger  $n_{\text{test}}$ .  
We need to hold back some data for testing, that's unavoidable. This makes GE-estimate tendentially larger than it could be because model fit during training is hold back a tiny bit.  
by design: Pessimistic Bias of GE, overestimates unknown true GE due to  $n_{\text{train}} < n$ !

Solution: **Resampling** techniques to properly estimate Learner GE. Reusing information from full dataset in each sampling iteration, randomness/Variance can cancel out ideally.  
We split s.t.  $n_{\text{test}}$  is sufficiently large. Majority of observations (typically 80-90%) are used for training.  
Good aggregate Performance

**Procedure with Resampling** s-times resampling of dataset, so we have  $S$  split iterations

- 1) In each iteration find  $\hat{f}_i$  through ERM on  $\mathcal{D}_{\text{train}}^{(i)}$ . These models are just intermediate results, model-type restricted by Inducer  $\Rightarrow$  use this specific learner
- 2) In each iteration calculate  $GE_i$  with our chosen outer loss on  $\hat{f}_i$  predictions in  $\mathcal{D}_{\text{test}}^{(i)}$  (hyperparameters) to fit final model  $\hat{f}$
- 3) Estimate  $\hat{GE} = \frac{1}{S} \sum_{i=1}^S GE_i$  as average GE across all resampling iterations
- 4) Repeat this for many different learners/learner configs on all data

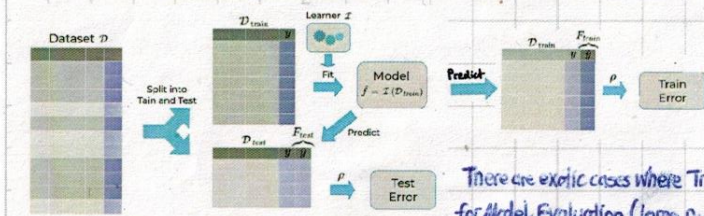
# Training and Test Error

## Training Error and Test Error

Let  $\hat{f} = I(D_{\text{train}}, \lambda)$  and let  $\rho$  be a performance metric. We define the

- **Training Error** as  $\rho(y_{\text{train}}, \hat{f}_{D_{\text{train}}}) \triangleq \text{Remp}(\hat{f})$  when we use same  $\rho$  metric as inner loss used
- **Test Error** as  $\rho(y_{\text{test}}, \hat{f}_{D_{\text{test}}})$  estimator for GE of model/learner, decides model choice

! Choose  $D_{\text{test}}$  and  $D_{\text{train}}$ , s.t.  $D_{\text{test}} \cap D_{\text{train}} = \emptyset$  to avoid optimistic Bias.



There are exotic cases where Train Error can be used approx. for Model Evaluation (large  $n_{\text{train}}$  with weird spiral data example)

## Train Error vs. Test Error

Goodness-of-fit measures - like  $R^2$ ,  $\chi^2$ , AIC, BIC, deviance - is based on training error but are based on (might work for models of restricted capacity where certain assumptions re true) distributional assumptions and large enough data - therefore hard to use for high-dimensional or more complex data.

- Decrease of  $n_{\text{train}}$   $\Rightarrow$  Increase of Test Error (in general) b.c. model generalises better with more training data and worse with less training data.
- Increase of complexity  $\Rightarrow$  Decrease of Training Error (in general) b.c. it becomes easier to learn all patterns on small training data sets or with more flexibility, but overfitting risk rises.
- Decrease of  $n_{\text{test}}$   $\Rightarrow$  Increase of Variance of test error. (decrease of Var (Test error), more reliable)

## Bias - Variance - Tradeoff

The GE of the inducer can be estimated by the Test Error.

The quality of this estimate is based on the Bias(s) and the Variance of the Test Error. (1) Smaller Test Error means less Bias.

Those are influenced by the complexity and amount of training data.

! Inference Theory:  $MSE(\hat{GE}, GE) = \text{Var}(\hat{GE}) + \text{Bias}(\hat{GE}, GE) + \text{Bayes-Error}$   
 $\text{Bias}(\hat{GE}, GE) = E(\hat{GE}) - GE$

! Captures everything unknown that we can't include in our modeling.  
 Bayes Error] irreducible error due to inherent randomness, noise... (occurs even with perfect knowledge)

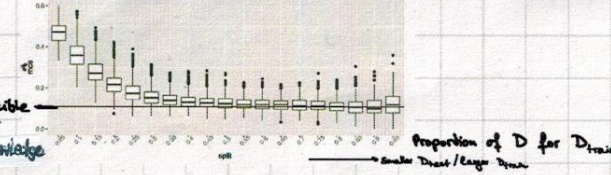
Lowest possible prediction error of optimal Learner given a certain perf. measure/loss function

Hold-out sampling produces Bias-Variance-Tradeoff that is controlled by split ratio

Influence of  $n_{\text{train}}$ :

Because in practice  $n = |D|$  is fixed, increase/decrease of  $n_{\text{train}}$  results in a decrease/increase of  $n_{\text{test}}$ . So

Increasing  $n_{\text{train}}$  leads to decrease in Test Error but higher Variance of Test Error.  
 Decreasing  $n_{\text{train}}$  leads to worse fit, therefore higher Test Error due to stronger pessimistic bias in GE



Rule of Thumb for hold-out split: But in practice we should not use hold-outs, esp. for small  $n$ , and instead apply resampling!

Model influenced by: Learner Algorithm, chosen Loss functions, data, randomness

OF and UF are (theoretical) properties of ERM-optimal fixed models  $\hat{f}$  that we should never apply

## Overfitting and Underfitting

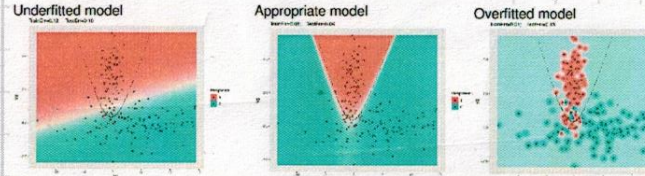
There is an overlap between UF and OF, no clear cutoff (terrible models do both)

**Underfitting** occurs when a model can't reflect the true shape of underlying function (given the data)

$\Rightarrow$  High Test Error and high Training Error.  $UF(\hat{f}; L) := GE(\hat{f}; L) - GE(f^*; L)$   $f^*$  is unknown Bayes optimal model

**Overfitting** occurs when a model reflects noise or artifacts from  $D_{\text{train}}$  which do not generalise.

$\Rightarrow$  Small Train Error and high Test Error.  $OF(\hat{f}; L) := GE(\hat{f}; L) - \text{Remp}(\hat{f}; L)$  in practice  $OF(\hat{f}; L) > 0 \forall \hat{f}$  but problem starts when



Underfitting hard to detect as  $f^*$  unknown (can only estimate it with very large  $n$  extra)  
 How to reduce Train & Test Error if we don't know correct Learner (conflict)  
 In this example: We don't know Bayes error rate of optimal classifier

technically also underfitted [in these cases OF more dominant problem, has "prio"]

Overfitting is influenced by

- Complexity of Model, i.e.  $\dim(\Theta)$
- Amount of Training Data, i.e.  $n_{\text{train}}$
- Dimensionality of Feature Space, i.e.  $\dim(X)$
- Irreducible Noise, i.e. aleatoric uncertainty

The less data we have and the more features we want to include, the more we should lean towards low complexity models, and vice versa.  
 More data  $\Rightarrow$  less randomness that complex model can overfit on.

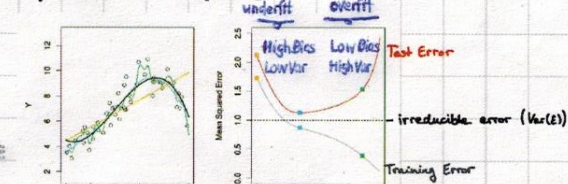
Overfitting can be avoided by regularization methods:

- Constrain  $\mathcal{H}$  by using less complex model classes or Hyperparameter Tweaking
- Higher amount or better quality of training data
- Use Feature Selection/Engineering and keep only the "Early Stopping" features that carry a lot of useful information.
- Occam's razor: If two models have similar GE, prefer the simpler one.

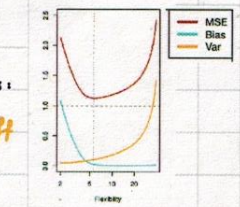
although it's possible that ERM result is simple LM and higher-degree coeff. all = 0

Polynomial Learner] degree  $\uparrow \uparrow$  will lead to interpolation-like overfit  
 This explains one big problem of Interpolators like Interpol. splines or Interpol. Gaussian processes whose predictions always perfectly match regression target, but interpolate random noise as well.

Influence of complexity:



We can deconstruct the MSE of the Test Error into it's Variance and Bias:  
 We can see that we need to trade-off in order to minimize the MSE.



# Resampling



Looks like a dumpling which rhymes with Resampling

## Setup

We want to estimate  $GE(\mathbb{E}, \lambda, n, p_L) = GE(\hat{f}, L)$

but doing so with a single hold-out-split results in a high pessimistic Bias & high Variance of  $\hat{GE}$

So instead we split repeatedly and average the result

• This way we reduce the variance from small test sets.  
• also reduce pessimistic bias of  $\hat{GE}$  with larger split ratio  $\frac{|D_{test}|}{|D|}$  which resampling indirectly enables us to do by  $(n_{train})_i$  & working out because of resampling process averaging

## Resampling Strategies

We choose  $J_{train} \in \{1, \dots, n\}^{n_{train}}$  and  $J_{test} \in \{1, \dots, n\}^{n_{test}}$

"Choose  $n_{train}$  indices from  $1, \dots, n$  with replacing"

We define  $J := (J_{train,1}, J_{test,1}), \dots, (J_{train,B}, J_{test,B})$

with  $B \in \mathbb{N}$  and  $n_{train,1} \approx \dots \approx n_{train,B} \approx n_{train}$

Our estimate is then aggregation overall, usually  $mean()$

$$\hat{GE}(I, \lambda, J, p_L) = \frac{1}{B} \sum_{k=1}^B p(y_{J_{test,k}}, F_{J_{train,k}, I(D_{train,k}, \lambda)})$$

Typically, we re-use exact splits for learner(config) comparisons, comparable

reproducible

## Bias-Variance Analysis in Resampling

If there exists a dedicated Test set that is not used to train  $\hat{f}$ , then  $\hat{GE}(\hat{f}, L) = \frac{1}{m} \sum_{i=1}^m L(y_i, \hat{f}(x_i))$  is unbiased!

with the  $m$  samples being i.i.d. and  $E[\hat{GE}(\hat{f}, L)] = E[L(y, \hat{f}(x))] = GE(\hat{f}, L)$  and  $V[\hat{GE}(\hat{f}, L)] = \frac{1}{m} V[L(y, \hat{f}(x))]$

We can apply the Central Limit Theorem to approx. the distr. of  $\hat{GE}(\hat{f}, L)$  and compute Confidence Intervals and hyp. tests

If  $\hat{f}$  is trained on  $D$  we estimate  $GE(I, \lambda, n, p)$  instead of  $GE(\hat{f}, L)$  using a resampling based estimate  $\hat{GE}(I, \lambda, J, p)$

! So our estimate for  $GE(I, \lambda, n, p)$  is in expectation nearly correct, i.e.  $n_{train}$  instead of  $n$ . b.c.  $n_{train} < n$  our estimate is pessimistically biased - on  $n$  it would perform better

! Holdout Subsampling: technically also overlap (for random, not systematic) but corr. of  $\hat{GE}$  can't be proven, and corrected for

## Subsampling/Monte Carlo CV

1. Split the data into two partitions
  2. Use one set for training and the other for testing. → Results in one test error
  3. Repeat step 1. & 2. many times
  4. Average over all test errors
- !  $\frac{2}{10}$  or  $\frac{1}{10}$  of the data for training is common.

## Bootstrap

1. Draw a bootstrap of size  $|D|$  with replacement from the full data set  $D$ . The data of  $D$  that is not drawn is the test set.
  2. Train and evaluate the model → Results in a test error
  3. Repeat 1. & 2. many times
  4. Average over all test errors
- ! A training set contains about 3/4 unique points (for  $n \rightarrow \infty$ )  
 $1 - \mathbb{P}((x, y) \in D_{train}) = 1 - (1 - 1/n)^n \xrightarrow{n \rightarrow \infty} 1 - 1/e \approx 63.2\%$

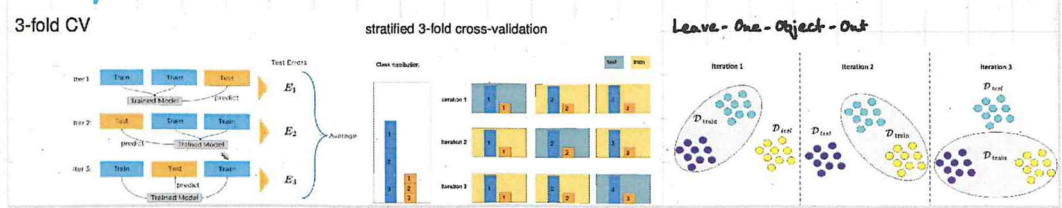
## Cross Validation

1. Split the data into  $k$  equally-sized partitions. Fraction  $\frac{k-1}{k}$  for train,  $\frac{1}{k}$  for test
  2. Each set is test set once, the rest is used for training → Results in  $k$  test errors (and training errors)
  3. Average the test errors
- ! 5 or 10 fold CV's are common.  $n$ -fold-CV is also called Leave-one-out-CV → Nearly unbiased but high variance
- ! The test errors are not independent b.c. the splits aren't independent → There is no unbiased estimate for  $V(\hat{GE}(\dots))$ .  
Esp. for very small  $n$  we should consider integrating Monte Carlo approach into CV ⇒ Repeat CV with large  $k$  and average over it to reduce  $Var(\hat{GE})$

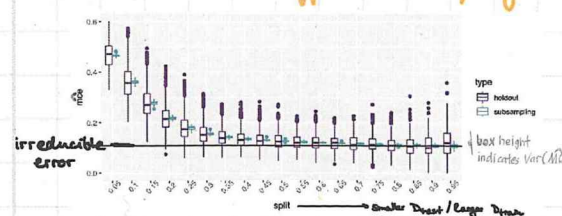
## Leave-one-Object-Out

This method is used when there are multiple observations per object → Data is not i.i.d. anymore.  
The data of one object should either be in the training set or test set! Not in both → Otherwise  $\hat{GE}$  is biased.  
This can be achieved by using CV on objects, i.e. split the objects into  $k$  groups...

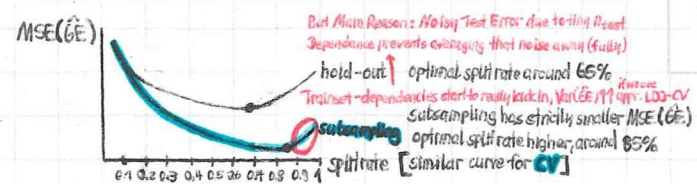
## Example CV



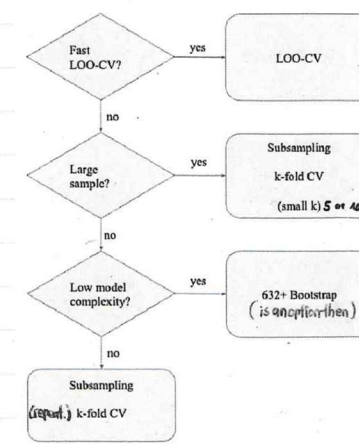
## Bias-Variance-Trade-off in Subsampling



!  $Bias(\text{Subsampling} | \text{split-rate}) \neq Bias(\text{Hold-out} | \text{split-rate})$   
 $Var(\text{Subsampling} | \text{split-rate}) < Var(\text{Hold-out} | \text{split-rate})$   
⇒ "optimal" split-rate is higher in Subsampling compared to one hold-out-split.



## Guidelines



Also watch out for class effects and consider stratification! (esp. with simple holdout split)  
Even large datasets with  $n \geq 100,000$  can have "hidden" small sample sizes, e.g. one group very small.

# Performance Measures for Linear Regression

Pointwise outer loss  $|D_{\text{test}}| = m$

Mean squared error (MSE):  $f_{\text{MSE}}(\hat{y}, \mathcal{T}) = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2 \in [0, \infty)$  Root MSE (RMSE)  $\rightarrow f_{\text{RMSE}}(\hat{y}, \mathcal{T}) = \sqrt{f_{\text{MSE}}}$   
back to original scale

Sum of squared errors (SSE):  $f_{\text{SSE}} = m \cdot f_{\text{MSE}}$  very similar

- MSE generalizes L2 loss to performance metric. Then we use equivalent inner & outer loss
- Outliers with large prediction error heavily increase MSE as they enter quadratically

Mean absolute error (MAE):  $f_{\text{MAE}} = \frac{1}{m} \sum_{i=1}^m |y^{(i)} - \hat{y}^{(i)}| \in [0, \infty)$  generalizes L1 loss

Even more robust against outliers is Median Absolute Error  $f_{\text{MedAE}} = \text{Med}(|y^{(i)} - \hat{y}^{(i)}|)$

Mean absolute percentage error (MAPE):  $f_{\text{MAPE}} = \frac{1}{m} \sum_{i=1}^m \left| \frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}} \right| \in [0, \infty)$

- Small  $|y|$  have more influence; can't handle  $y=0$

Mean absolute scaled error (MASE):  $f_{\text{MASE}} = \frac{f_{\text{MAE}}}{\text{"Naive MAE"}}$  with  $\text{MAE}_{\text{naive}} = \frac{1}{m-1} \sum_{i=1}^m |y^{(i)} - y^{(i-1)}|$  naive forecast

- scale independent error ratio, used for time series forecasts
- Aim for  $\text{MASE} < 1$ , forecast more accurate than naive forecast (on average)

## Set-based outer Loss

$R^2: f_{R^2}(\hat{y}, \mathcal{T}) = 1 - \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^m (y^{(i)} - \bar{y})^2} = 1 - \frac{\text{SSE}}{\text{SST}}$  "Fraction of  $y$ -variance explained" by the model  
More specifically the influence of features on target  
 $\frac{\text{SSM}}{\text{SST}} = \frac{\sum (\hat{y}^{(i)} - \bar{y})^2}{\sum (y^{(i)} - \bar{y})^2}$   
 $= 1 - \frac{f_{\text{MSE}}(\hat{y}, \text{LM}(\hat{y}))}{f_{\text{MSE}}(\hat{y}, \bar{y})} \hat{=} 1 - \frac{\text{SSE}_{\text{LM}}}{\text{SSE}_{\text{const. Model}}}$  - simple model just using intercept, constant prediction  $\bar{y}$  (mean)

SSE reduction of complex model vs. constant model baseline, generalized to ML with outer Loss:  $R^2 = 1 - \frac{\text{Loss}_{\text{complex Model}}}{\text{Loss}_{\text{simplest Model}}}$  for model comparisons beyond LM even different model types

In Linear Modeling  $R^2$  is typically only used for evaluating on train data where  $R^2 \in [0, 1]$  always, can't be worse than const.  $\bar{y}$ -prediction  
 $R^2 = 1$ : perfect prediction  $R^2 = 0.9$ : LM reduces SSE / Loss by factor 10  $R^2 = 0$ : Predict as badly as const. model

On test set  $R^2 < 0$  is possible because LM not fitted to new data, Quadr. summenzerlegung ungültig  $\Rightarrow$   $\text{SSE}_{\text{LM}}$  can be  $> \text{SSE}_{\text{const. Model}}$   
strong indicator for terrible underfitted model or highly overfitted model that generalizes horrendously to new unseen data!

- Higher  $R^2$  does not always equate better fit, not even on training data

$R^2$  can simply be increased by using more data (wider range) while fit measured with MSE stays the same or even worse (less data  $\rightarrow$  easier to fit patterns)

$R^2$  increases with more features used and higher complexity of model, but only on train data. For new data this makes overfitting more likely

✓  $R^2$  invariant w.r.t. linear scaling of  $y$  which is not true for MSE

Spearman Correlation (rank-based):  $f_{\text{Spearman}}(\hat{y}, \mathcal{T}) = \frac{\text{Cov}(\text{rg}(y), \text{rg}(\hat{y}))}{\sqrt{\text{Var}(\text{rg}(y)) \text{Var}(\text{rg}(\hat{y}))}} \in [-1, 1]$

- Very robust against outliers, don't care about actual values just ranks
- Invariant under monotone transformations of  $y$

$|f_{\text{Spearman}}| = 1$ , so  $f_{\text{Sp}} \in [-1, 1]$  means perfect monotonic relationship between  $y$  and predictions  $\hat{y}$

# Measures for Classification

## Pointwise outer Loss using class Labels

Missclassification Error Rate (MCE):  $p_{MCE} = \frac{1}{m} \sum_{i=1}^m \mathbb{I}[y^{(i)} \neq \hat{y}^{(i)}] = \frac{Z_{\text{False pos}} + Z_{\text{False neg}}}{\text{Total observations}} \in [0,1]$   $p_{MCE} = 1 - p_{\text{Acc}}$

Ind.  $y_i \in \{0,1\}$  could also call this a 0-1 Loss-function  $L_{0-1}(y_i, \hat{y}_i) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}[y_i \neq \hat{y}_i]$

Accuracy (ACC):  $p_{\text{Acc}} = \frac{1}{m} \sum_{i=1}^m \mathbb{I}[y^{(i)} = \hat{y}^{(i)}] = \frac{Z_{\text{True Pos}} + Z_{\text{True Neg}}}{\text{Total Observations}} \in [0,1]$  "Proportion of correctly classified observations"  $p_{\text{Acc}} = 1 - p_{MCE}$

ACC technically not a loss cause we aim to max it  $\Rightarrow$  Minimizing MCE

Problems: MCE says nothing about how good/skewed the predicted probabilities are. Accuracy is bad performance measure when Label distr. is unbalanced, for example if only tiny fraction of 0.05% have certain disease.  $\rightarrow$  Always predicting "No disease" ( $\hat{y}=0$ ) has ACC=0.9995 and MCE=0.0005. Terrible system that sends all sick patients home. "Accuracy Paradox"

Errors on all classes are weighted equally which is often inappropriate!  
Example disease prediction: False Negative might be much worse than False Positive!  
"Imbalanced Cost Problem"

## Pointwise outer Loss using probabilities

Brier Score (BS):  $p_{BS} = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$  MSE for probabilities encoded as 0/1 in binary case

Multiclass Brier:  $p_{BS,MC} = \frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K (\sigma_k^{(i)} - \hat{\pi}_k^{(i)})^2$   $C_{y^{(i)}=k}$  One-hot-encoded class label

Log/Bernoulli-Loss (LL):  $p_{LL} = \frac{1}{m} \sum_{i=1}^m (-y^{(i)} \log(\hat{y}^{(i)}) - (1-y^{(i)}) \log(1-\hat{y}^{(i)}))$

$\uparrow$  Logistic Regression

Multiclass:  $p_{LL,MC} = \frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \sigma_k^{(i)} \log(\hat{\pi}_k^{(i)})$

Both Brier-Score and Log Loss: Optimal value is 0, "confidently wrong" predictions are penalized (with Log Loss much more heavily at the edges)

Brier Score for individual obs.  $\in [0,1]$  so capped at 1. Log-Loss  $\in [0, \infty) \Rightarrow$  tries really hard to avoid False Pos & False Neg. Predictions with high  $\hat{y}^{(i)}$  with low  $\hat{y}^{(i)}$

## Confusion Matrix

		True classes			
		A	B	C	error
Multiclass:	Predicted Class A	50	0	0	(0) 50
	Predicted Class B	0	46	4	(4) 50
	Predicted Class C	0	4	46	(4) 50
	error	(0)	(4)	(4)	[8] -
	$\eta_{\text{true}}$	50	50	50	- 150

cell value = number of class k predictions for given True Class

		True Class $\eta$		
		$\eta$ pos.	$\eta$ neg.	
Pred. Class $\hat{\eta}$	$\hat{\eta}$ pos.	True Positive (TP)	False Positive (FP)	# Pos. Pred.
	$\hat{\eta}$ neg.	False Negative (FN)	True Negative (TN)	# Neg. Pred.
		# Pos. Class	# Neg. Class	$\eta$

Cost Matrix: Basically another Matrix with same dimensions on top of Confusion Matrix

Usually only #FN, #FP relevant for cost calculation (Usually true predictions get 0 entry in cost matrix)

Assign different costs to different errors of Confusion Matrix:  $\text{Costs} = \frac{1}{n} \sum_{i=1}^n C[y^{(i)}, \hat{y}^{(i)}]$  then costs =  $\frac{1}{n} \text{tr}((\text{Conf.mat})^T (\text{Cost.mat}))$

(for certain classes)

This allows us to put different weights on False Positive/Negative or focus on one error type entirely and setting all other cost matrix entries to zero!

However: in practice how to specify costs precisely? We could evaluate "from different perspectives, with multiple metrics, to get an idea of system quality. An important subfield of ML is cost-sensitive learning

## Set based outer Loss / ROC-Analysis

Many different metrics based on Confusion Matrix. Metric Focus Choice requires domain knowledge!