LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Evaluation: Introduction and Remarks

Department of Statistics – LMU Munich

INTRODUCTION

In predictive modeling, performance estimation can have different goals:

- **Performance estimation of a model:** Estimate *generalization* error of a model on new (unseen) data, drawn from the same data generating process that training data came from.
- **Performance estimation of an algorithm:** Estimate *generalization* error of a learning algorithm, trained on a data set of a certain size, on new (unseen) data, all drawn from the same data generating process.
- **Model selection:** Select the best model from a set of potential candidate models (e.g., different model classes, different hyperparameter settings, different features)
- **Learning curves:** How does the generalization error scale when an algorithm is trained on training sets of different sizes?

Obviously, all goals are quite related, i.e., reliable estimation of (predictive) performance is the foundation for all of them.

PERFORMANCE ESTIMATION

- Goal: Estimate performance on new data
- For now, we start by assuming to have a fixed model, already fitted on some data.
- We also assume to have some reasonable data for testing available.
- ML performance evaluation provides clear and simple protocols for reliable model validation. These protocols are often much simpler than classical statistical model diagnosis and rely only on few assumptions
- Most important assumption: Data we use is realistic and i.i.d.
- ML evaluation is still hard enough and offers LOTS of options to cheat yourself and especially your clients, mistakes can happen on many levels.

PERFORMANCE ESTIMATION

Clearly, we are looking for a statistical estimator - for the generalization error! Different levels of randomness are involved:

- Even if we are evaluating on a fixed test data set, this is only a sample and not full reality. The sample can be too small, then our estimator will be of high variance; the sample could not be from the distribution of interest, then our estimator will be biased.
- The same holds true for our model - it was only fitted on a sample. This creates another source of randomness, the training data sample. This is true, even if the fitting algorithm is deterministic.
- In ML, many learning algorithms are stochastic: Think random forest, or stochastic gradient descent. This is a third source of randomness.

METRICS: INNER VS. OUTER LOSS

- To judge the performance of a model on a given data set, we might want to produce a quantitative measure of the performance on that set.
- Usually we define a function that measures the quality of a prediction per observation.
- We then aggregate over the complete set - often by some form of averaging.

Don't we already know this? Sounds like loss functions and risk estimation? It nearly is the same. Nearly!

METRICS: INNER LOSS

- We already covered this, its associated empirical risk is optimized during model fitting
- The keyword above is **optimization**, some functions are much tamer to handle than others, smoothness and differentiability are often required for efficient optimization
- For this reason, we often choose something that's easier to handle numerically
- Another pretty practical reason might be: Our toolkit of choice only implements the optimization of a certain inner loss; changing the outer loss to something custom is simple, changing the inner often is not

METRICS: OUTER LOSS

- Performance metric to assess the model
- Should be carefully considered and selected
- There are no objectively better or worse metrics - YOU have to select depending on your application, domain and what you want
- Except for “should be reasonable and reflect what I want” there are no huge requirements for an outer metric, it is a function which takes a vector of prediction and a vector of labels and evaluates them
- Think about what will happen after a (wrong) prediction of your model, that should help to design an outer loss
- Yes, in model selection (later), we optimize it, but we use special techniques there anyway that can deal with arbitrary metrics

METRICS: INNER VS. OUTER LOSS

Usually, it is desired that inner and outer loss match, however, this is not always possible, as the outer loss is often numerically hard(er) to be handled during optimization and we might opt to approximate it.
Examples:

- In logistic regression we minimize the binomial loss
- In kNN there is no explicit loss minimization
- But when evaluating the models we might be more interested in (cost-weighted) classification error
- Or some of the more advanced metrics from ROC analysis like AUC

Nowadays, one can also optimize many of the harder losses directly, but this is less standard, less often implemented and we will not cover this here.

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Evaluation: Simple Metrics for Regression and Classification

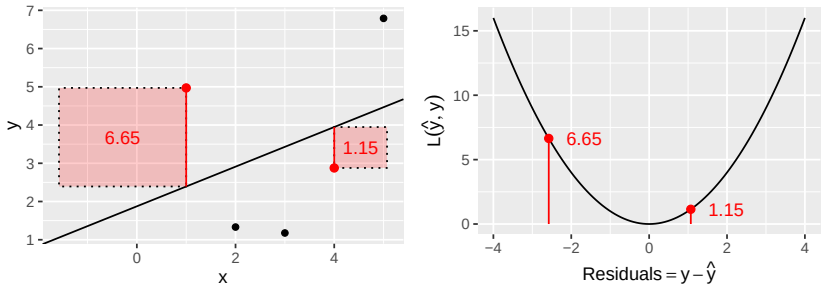
Department of Statistics – LMU Munich

REGRESSION: MSE

The **Mean Squared Error** compares the mean of the squared distances between the target variable y and the predicted target $\hat{y}$.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 \in [0; \infty]$$

Single observations with a large prediction error heavily influence the **MSE**, as they enter quadratically.



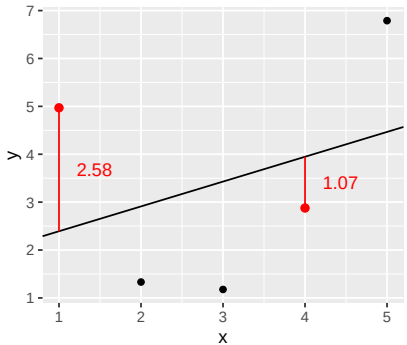
We could also sum the errors up (SSE), or take the root (RMSE) to bring the measurement back to the original scale of the outcome.

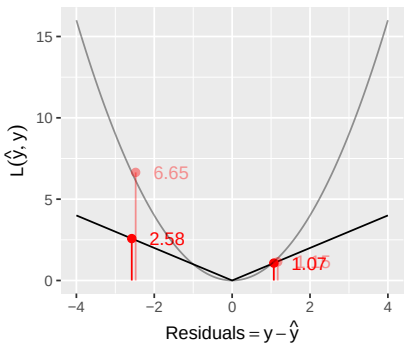
REGRESSION: MAE

A more robust (but not neccessarily better) way to compute a performance measure is the **Mean Absolute Error**:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y^{(i)} - \hat{y}^{(i)}| \in [0; \infty]$$

Less influenced by large errors and maybe more intuitive than the MSE.





Instead of averaging we might also consider the median for even more robustness.

LABELS: ACCURACY / MCE

The misclassification error rate (MCE) simply counts the number of incorrect predictions and presents them as a rate, accuracy is defined in a similar fashion for correct classifications

$$MCE = \frac{1}{n} \sum_{i=1}^{n} [y^{(i)} \neq \hat{y}^{(i)}] \in [0; 1]$$

$$ACC = \frac{1}{n} \sum_{i=1}^{n} [y^{(i)} = \hat{y}^{(i)}] \in [0; 1]$$

- If the data set is small this can be quite a brittle measure
- The MCE says nothing about how good or skewed predicted probabilities are
- Errors on all classes are weighed equally, that is often inappropriate

LABELS: CONFUSION MATRIX

Much better than simply reducing prediction errors to a simple number we can tabulate them in a confusion matrix, tabulating true classes in rows and predicted classes in columns. We can nicely see class sizes (predicted and true) and where errors occur.

```
##            setosa versicolor virginica -err.- -n-
## setosa         50          0         0      0  50
## versicolor      0         46         4      4  50
## virginica       0          4        46      4  50
## -err.-          0          4         4      8  NA
## -n-            50         50        50     NA 150
```

LABELS: COSTS

We can also assign different costs to different errors via a cost matrix.

$$Costs = \frac{1}{n} \sum_{i=1}^{n} C[y^{(i)}, \hat{y}^{(i)}]$$

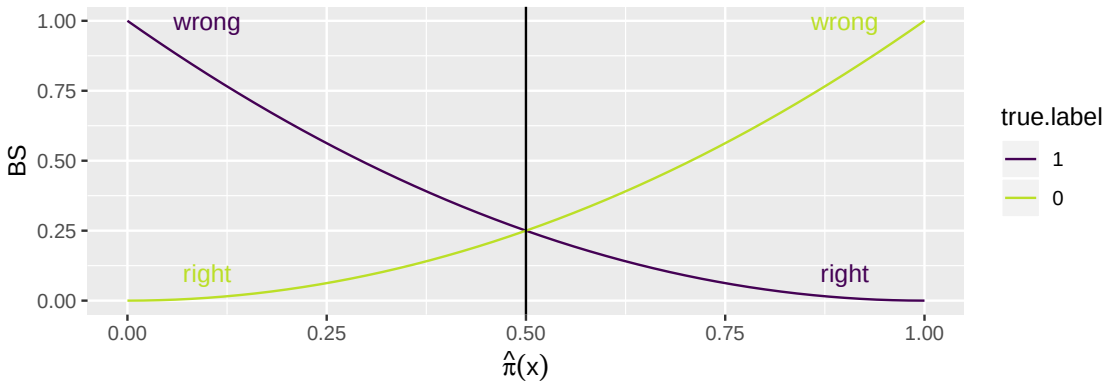
```
## Confusion matrix
##             predicted
## true         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         46         4
##   virginica       0          4        46
## Cost matrix C
##             predicted
## true         setosa versicolor virginica
##   setosa          0          1         1
##   versicolor      2          0         5
##   virginica       1          1         0
```

- Here, we penalize errors on class *versicolor* more heavily
- $Costs = (3 \cdot 5 + 4 \cdot 1)/150$

PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

$$BS1 = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{\pi}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

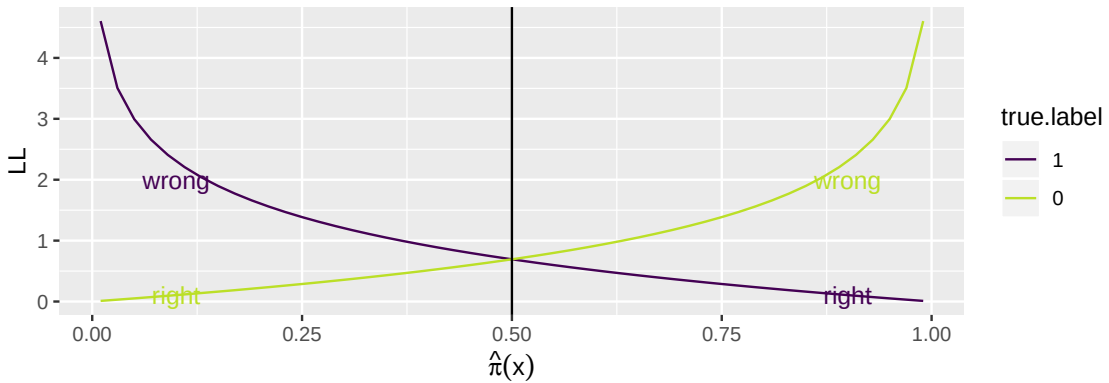


- Usual definition for binary case, $y^{(i)}$ must be coded as 0 and 1.
- Fancy name for MSE on probabilities

PROBABILITIES: LOG-LOSS

Logistic regression loss function, a.k.a. Bernoulli or binomial loss, $y^{(i)}$ coded as 0 and 1.

$$LL = \frac{1}{n} \sum_{i=1}^{n} \left(-y^{(i)} \log(\hat{\pi}(\mathbf{x}^{(i)})) - (1 - y^{(i)}) \log(1 - \hat{\pi}(\mathbf{x}^{(i)})) \right)$$



- Optimal value is 0, “confidently wrong” is penalized heavily
- Multiclass version: $LL = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{g} o_k^{(i)} \log(\hat{\pi}_k(\mathbf{x}^{(i)}))$

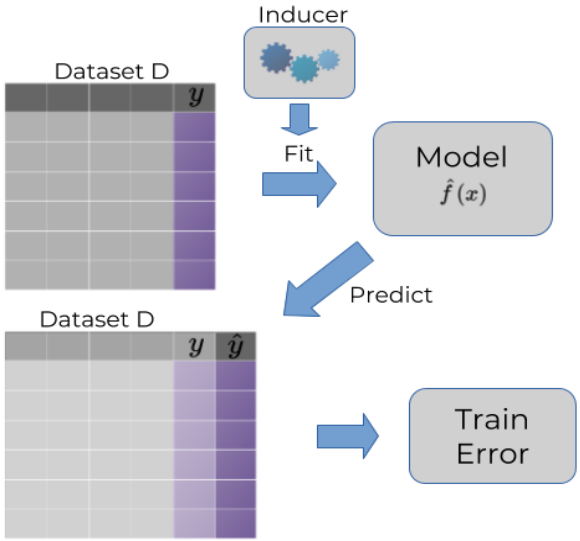
Introduction to Machine Learning

Evaluation: Train and Test Error

Department of Statistics – LMU Munich

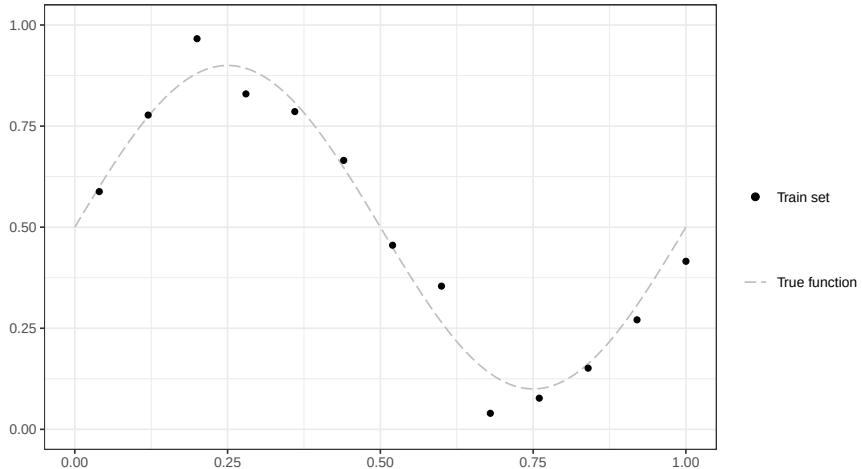
TRAINING ERROR

The *training error* (also called apparent error or resubstitution error) is estimated by the averaging error over the same data set we fitted on:



EXAMPLE: POLYNOMIAL REGRESSION

Assume an (unknown) sinusoidal function that $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$ that we sample from with some measurement error ϵ.

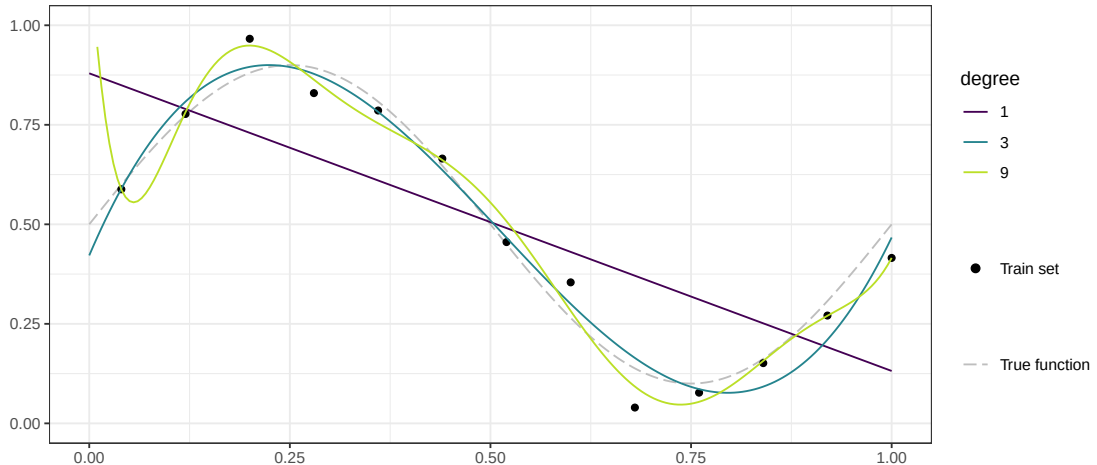


We try to approximate it with a dth-degree polynomial

$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^{d} \theta_j x^j.$$

EXAMPLE: POLYNOMIAL REGRESSION

Models of different *complexity*, i.e., of different orders of the polynomial are fitted. How should we choose d?



- d=1: 0.036: Clear underfitting
- d=3: 0.003: Pretty OK?
- d=9: 0.001: Clear overfitting

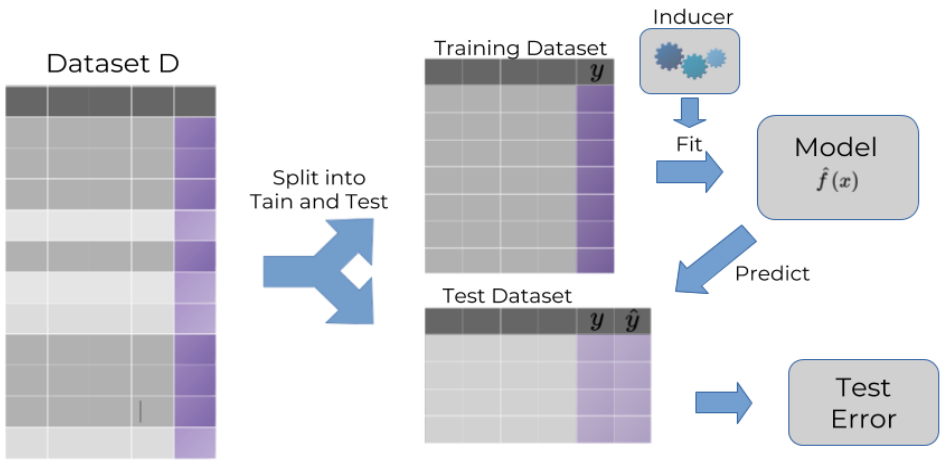
Simply using the training error seems to be a bad idea.

TEST ERROR AND HOLD-OUT SPLITTING

- The fundamental idea behind test error estimation (and everything that will follow) is quite simple
- To measure performance, let's simulate how our model will be applied on new, unseen data
- So, to evaluate a given model do exactly that, predict only on data not used during training and measure performance there
- That implies that for a given dataset $\mathcal{D}$, we have to preserve some data for testing that we cannot use for training

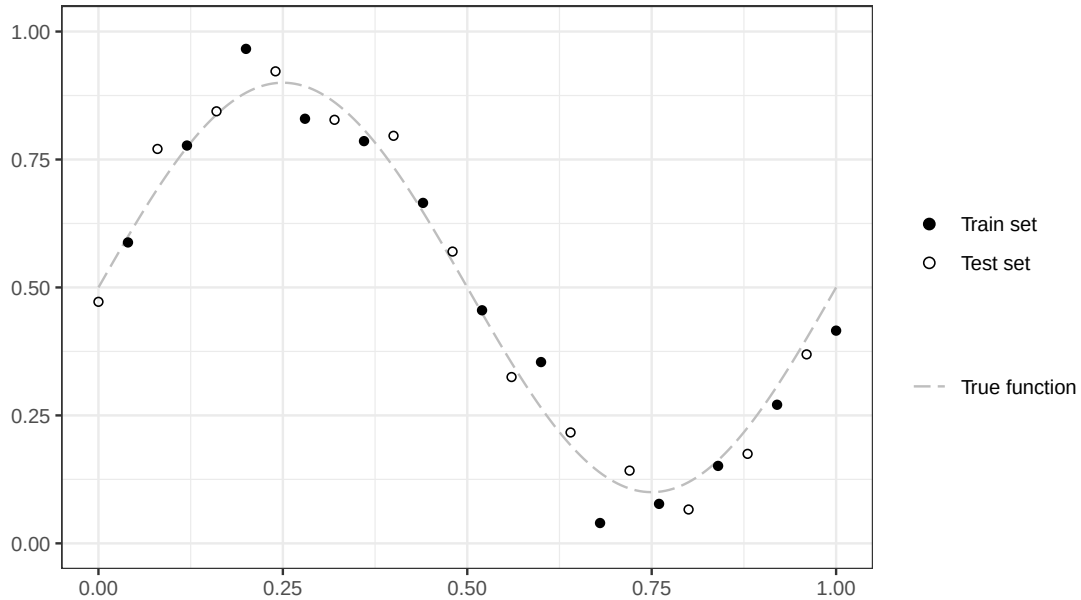
TEST ERROR AND HOLD-OUT SPLITTING

- Split data into 2 parts, e.g. a common setup is 2/3 for training, 1/3 for testing
- Evaluate on data not used for model building, no way to “cheat”

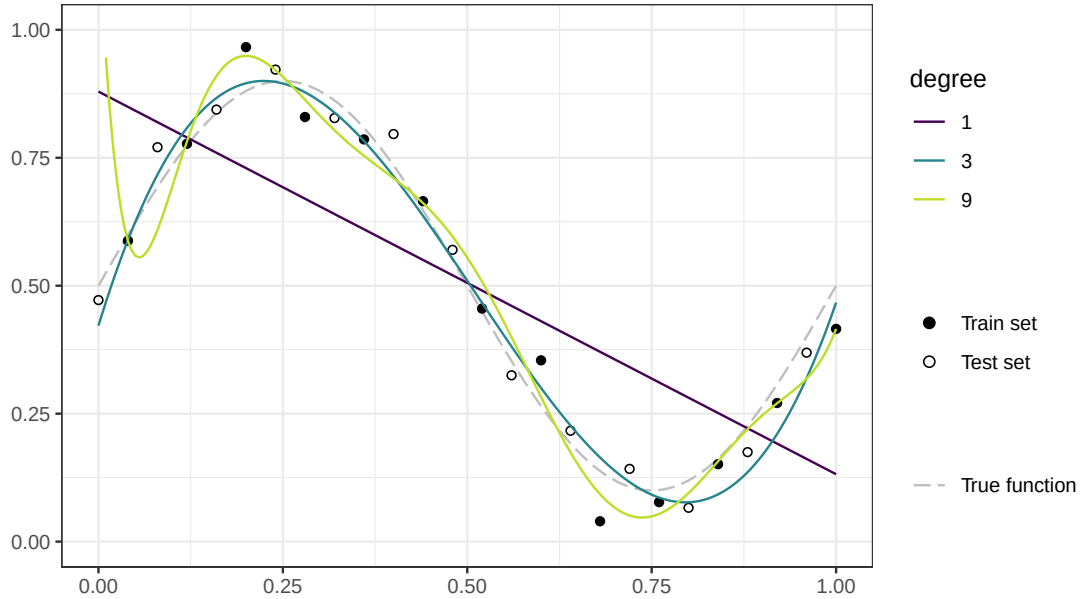


TEST ERROR AND HOLD-OUT SPLITTING

Let's consider some clean test data for our sinusoidal example:

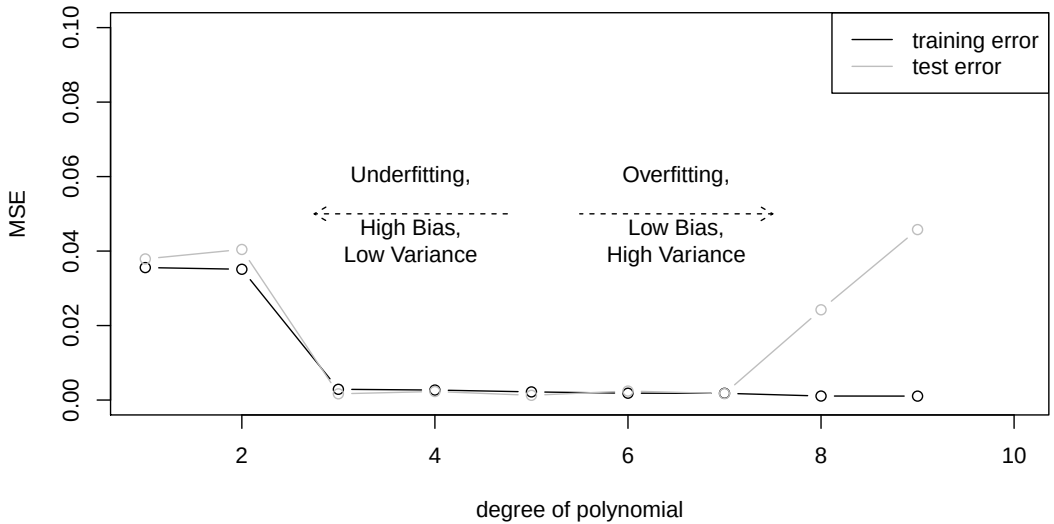


TEST ERROR AND HOLD-OUT SPLITTING



- d=1: 0.038: Clear underfitting
- d=3: 0.002: Pretty OK?
- d=9: 0.046: Clear overfitting

TEST ERROR AND HOLD-OUT SPLITTING



We can also plot error measures for all polynomial degrees. We see the common monotonous decrease in training error, if we increase model complexity (we can adapt better to data with more flexibility) and we see the common U-shape of the test error. First we underfit, then we over-fit, sweet-spot is in the middle. Numerically best for $d = 3$.

PROBLEMS OF TEST ERROR

A major point of confusion:

- In ML we are in a weird situation. We are usually given one data set. At the end of our model selection and evaluation process we will likely fit one model on exactly that complete data set. As training error evaluation does not work, we have now nothing left to evaluate exactly that model.
- Holdout splitting (and the soon following resampling) are tools just to estimate that future performance, to put that next to our final model. All of the models produced during that phase of evaluation are basically intermediate results.
- Keep that already in mind now, it will help to avoid confusion when we move on to cross-validation and nested cross-validation.

TRAINING VS. TEST ERROR

The training error

- is an over-optimistic (biased) estimator as the performance is measured on the same data the learned model was trained for
- decreases with smaller training set size as it is easier for the model to learn the underlying structure in the training set perfectly
- decreases with increasing model complexity as the model is able to learn more complex structures

The test error

- will typically decrease when the training set increases as the model generalizes better with more data (more data to learn)
- will have higher variance with decreasing test set size
- will have higher variance with increasing model complexity

Introduction to Machine Learning

Overfitting

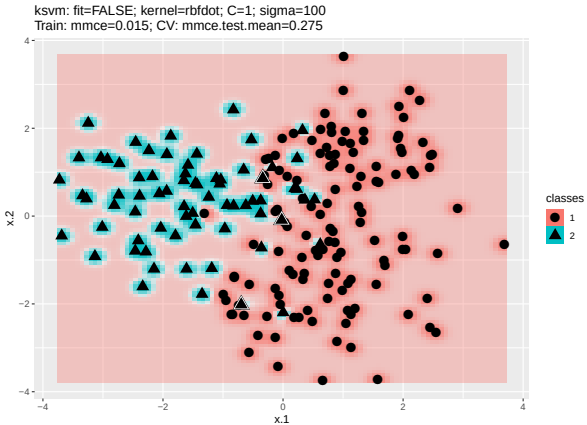
Department of Statistics – LMU Munich

OVERFITTING

- Overfitting is a well-known problem in ML for non-linear, powerful learning algorithms
- It happens when your algorithm starts modelling patterns in the data that are not actually true in the real world, e.g., noise or artefacts in the training data
- Happens when you have too many hypotheses and not enough data to tell them apart
- The more data, the more "bad" hypotheses are eliminated
- If the hypothesis space is not constrained, there may never be enough data
- There is often a parameter that allows you to constrain (*regularize*) the learner
- In this unit we will only give a very basic definition, and not really talk about measures against overfitting (see regularization!)

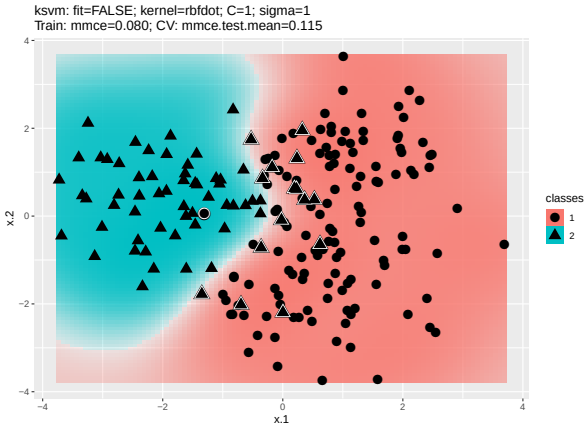
OVERFITTING

Overfitting learner



Better training set performance (seen examples)

Non-overfitting learner



Better test set performance (unseen examples)

OVERFITTING AND NOISE

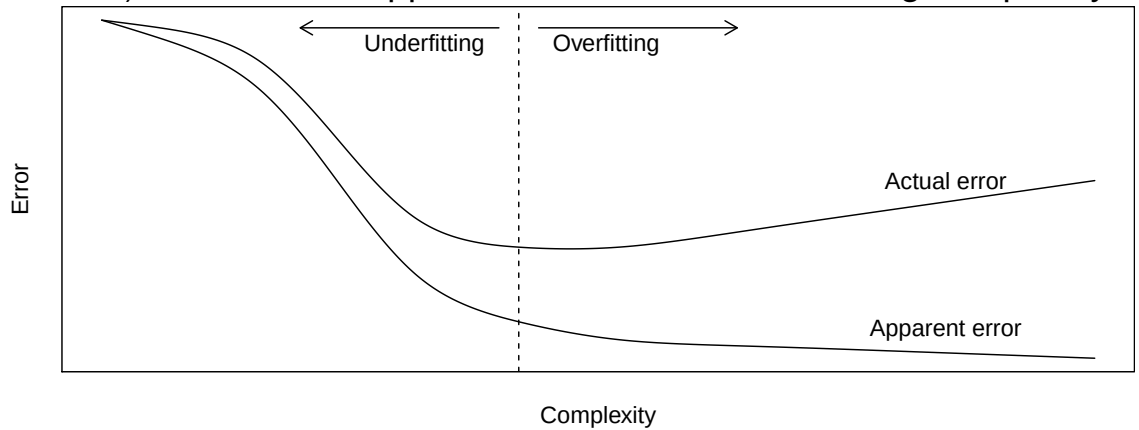
- Overfitting is seriously exacerbated by *noise* (errors in the training data)
- An unconstrained learner will start to model that noise
- It can also arise when relevant features are missing in the data
- In general it's better to make some mistakes on training data ("ignore some observations") than trying to get all correct

AVOIDING OVERFITTING

- You should never believe your model until you've *verified it on data that the learner didn't see*
- Scientific method applied to machine learning: model must make new predictions that can be experimentally verified
- Use less complex models
- Get more, or better data
- Some learner can do "early stopping" before perfectly fitting (i.e., overfitting) the training data
- Use regularization

TRADE-OFF BETWEEN GENERALIZATION ERROR AND COMPLEXITY

Apparent error (on the training data) and real error (prediction error on new data) evolve in the opposite direction with increasing complexity:



$\Rightarrow$ Optimization regarding the model complexity is desirable: Find the right amount of complexity for the given amount of data where generalization error becomes minimal.

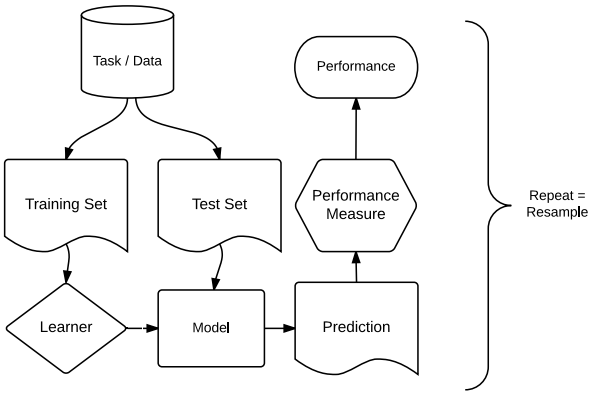
Introduction to Machine Learning

Evaluation: Resampling and Cross-Validation

Department of Statistics – LMU Munich

RESAMPLING

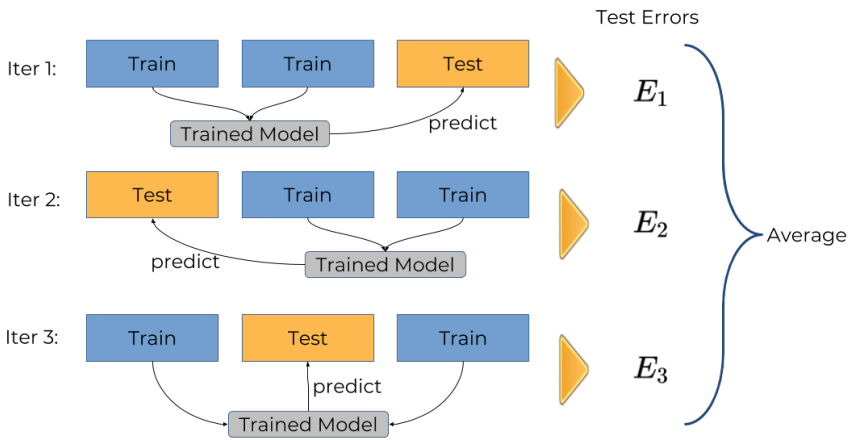
- Aim: Assess the performance of learning algorithm.
- Uses the data more efficiently then simple train-test.
- Repeatedly split in train and test, then average results.
- The usual trick is to make training sets quite larger (to keep the pessimistic bias small), and to handle the variance introduced by smaller test sets through many repetitions and averaging of results.



CROSS-VALIDATION

- Split the data into k roughly equally-sized partitions.
- Use each part once as test set and join the $k-1$ others for training
- Obtain k test errors and average.

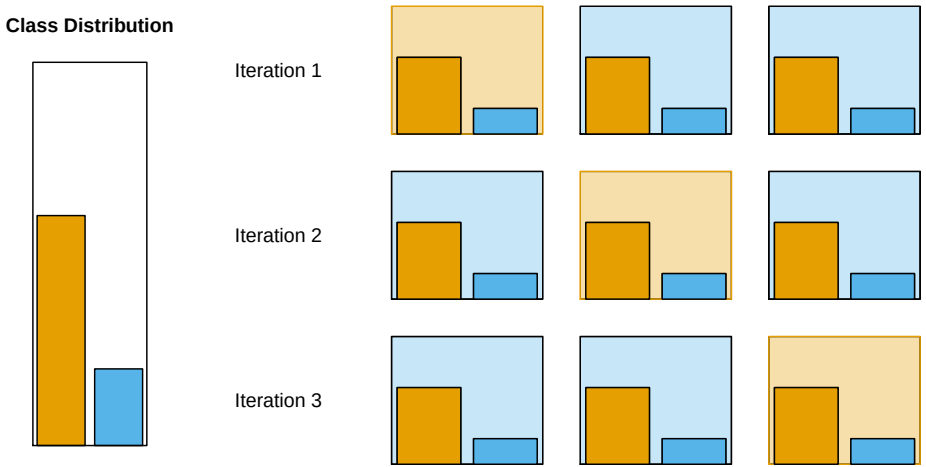
Example: 3-fold cross-validation:



CROSS-VALIDATION - STRATIFICATION

Stratification tries to keep the distribution of the target class (or any specific categorical feature of interest) in each fold.

Example of stratified 3-fold Cross-Validation:

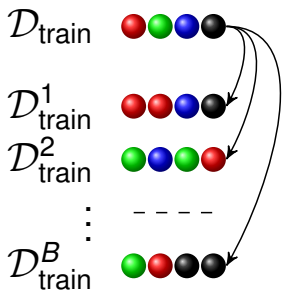


CROSS-VALIDATION - COMMENTS

- 5 or 10 folds are common, they use 80% and 90% of data in training
- $k = n$ is known as leave-one-out (LOO) cross-validation
- Estimates of the generalization error tend to be somewhat pessimistically biased (because the size of the training sets is $n - (n/k) < n$), bias increases as k gets smaller.
- The performance estimates for each fold are not independent, because of the structured overlap of the training sets. Hence, the variance of the estimator increases again for very large k (close to LOO), when training sets nearly completely overlap.
- LOO is nearly unbiased, but has high variance.
- Repeated k-fold CV (multiple random partitions) can improve error estimation for small sample size.

BOOTSTRAP

The basic idea is to randomly draw B training sets of size n with replacement from the original training set $\mathcal{D}_{\text{train}}$:



We define the test set in terms of out-of-bootstrap observations $\mathcal{D}^b_{\text{test}} = \mathcal{D}_{\text{train}} \setminus \mathcal{D}^b_{\text{train}}$.

BOOTSTRAP

- Typically, B is between 30 and 200.
- The variance of the bootstrap estimator tends to be smaller than the variance of k-fold CV, as training sets are independently drawn, discontinuities are smoothed out.
- The more iterations, the smaller the variance of the estimator.
- Tends to be pessimistically biased (because training sets contain only about 63.2% unique the observations).
- Bootstrapping framework might allow the use of formal inference methods (e.g. to detect significant performance differences between methods).
- Extensions exist for very small data sets, that also use the training error for estimation: B632 and B632+.

SUBSAMPLING

- Repeated hold-out with averaging, a.k.a. monte-carlo CV
- Similar to bootstrap, but draws without replacement, similar comments hold
- Typical choices for splitting: $4/5$ or $9/10$ for training
- The smaller the subsampling rate, the larger the pessimistic bias
- The more subsampling iterations, the smaller the variance

RESAMPLING DISCUSSION

- In ML we fit, at the end, a model on all our given data.
- Problem: We need to know how well this model performs in the future. But no data is left to reliably do this.
- In order to approximate this, we do the next best thing. We estimate how well the learner works when it sees nearly n points from the same data distribution.
- Holdout, CV, resampling estimate exactly this number. The "pessimistic bias" refers to when use much less data in fitting than n. Then we "hurt" our learner unfairly.
- Strictly speaking, resampling only produces one number, the performance estimator. It does NOT produce models, paramaters, etc. These are intermediate results and discarded.
- The model and parameters are obtained when we fit the learner finally on the complete data.
- This is a bit weird and complicated, but we have to live with this.

RESAMPLING DISCUSSION

- 5CV or 10CV have become standard
- Do not use Hold-Out, CV with few iterations, or subsampling with a low subsampling rate for small samples, since this can cause the estimator to be extremely biased, with large variance.
- For small data situation with less than 500 or 200 observations, use LOO or probably better repeated CV
- A $\mathcal{D}$ with $|\mathcal{D}| = 100.000$ can have small sample size properties if one class has only 100 observations ...
- For some models, computationally fast calculations or approximations for the LOO exist
- Modern results seem to indicate that subsampling has somewhat better properties than bootstrapping. The repeated observations can can cause problems in training algorithms, especially in nested setups where the "training" set is split up again.

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Evaluation: ROC Analysis 1

Department of Statistics – LMU Munich

IMBALANCED BINARY LABELS

- Consider a binary classifier for diagnosing a serious medical condition
- Here, label distribution is often **imbalanced**, i.e, not many people have the disease
- Evaluating with error rate for imbalanced labels is often inappropriate
- Assume that only 0.5 % of 1000 patients have the disease
- Always returning "no disease" has an error rate of 0.5 %, which sounds good
- However, this sends all sick patients home, which is the worst possible system, even classifying everyone as "sick" might be better, depending on what happens next
- This problem is sometimes known as the **accuracy paradox**

ROC ANALYSIS

ROC Analysis – which stands for "receiver operating characteristics – is a subfield of ML which studies the evaluation of binary prediction systems.

Quoting Wikipdia:

"The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields and was soon introduced to psychology to account for perceptual detection of stimuli. ROC analysis since then has been used in medicine, radiology, biometrics, forecasting of natural hazards, meteorology, model performance assessment, and other areas for many decades and is increasingly used in machine learning and data mining research"

CONFUSION MATRIX AND ROC METRICS

- From now on, we call one class "positive", one "negative" and their respective sizes n_+ and n_-
- The positive class is the more important, often smaller one
- We represent all predictions in a confusion matrix and count correct and incorrect class assignments
- False Positive means: We assigned "positive", but were wrong

		Actual Class y	
		Positive	Negative
$\hat{y}$ **Pred.**	Positive	**True positive** (TP)	**False positive** (FP, Type I error)
	Negative	**False negative** (FN, Type II error)	**True negative** (TN)

CONFUSION MATRIX AND ROC METRICS

We now normalize in rows and cols of the confusion matrix to derive several informative metrics in imbalanced or cost-sensitive situations:

		Actual Class y		
		Positive	Negative	
$\hat{y}$ **Pred.**	Positive	**True positive** (TP)	**False positive** (FP, Type I error)	Positive Predictive Value = $\frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}}$
	Negative	**False negative** (FN, Type II error)	**True negative** (TN)	Negative Predictive Value = $\frac{\#\text{TN}}{\#\text{FN} + \#\text{TN}}$
		True Positive Rate = $\frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}$	True Negative Rate = $\frac{\#\text{TN}}{\#\text{FP} + \#\text{TN}}$	Accuracy = $\frac{\#\text{TP} + \#\text{TN}}{\#\text{TOTAL}}$

- **TPR**: How many of the true 1s did we predict as 1?
- **TNR**: How many of the true 0s did we predict as 0?
- **PPV**: If we predict 1 how likely is it a true 1?
- **NPV**: If we predict 0 how likely is it a true 0?

EXAMPLE

		Actual Class y		
		Positive	Negative	
$\hat{y}$ Pred.	Positive	**True Positive** (TP) = 20	**False Positive** (FP) = 180	Positive predictive value = TP / (TP + FP) = 20 / (20 + 180) = **10%**
	Negative	**False Negative** (FN) = 10	**True Negative** (TN) = 1820	Negative predictive value = TN / (FN + TN) = 1820 / (10 + 1820) ≈ **99.5%**
		True Positive Rate = TP / (TP + FN) = 20 / (20 + 10) ≈ **67%**	True Negative Rate = TN / (FP + TN) = 1820 / (180 + 1820) = **91%**	

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

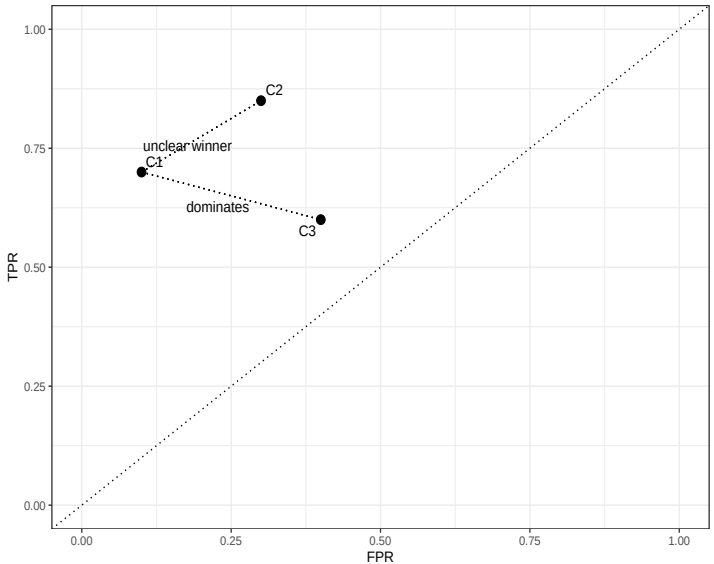
Introduction to Machine Learning

Evaluation: ROC Analysis 2

Department of Statistics – LMU Munich

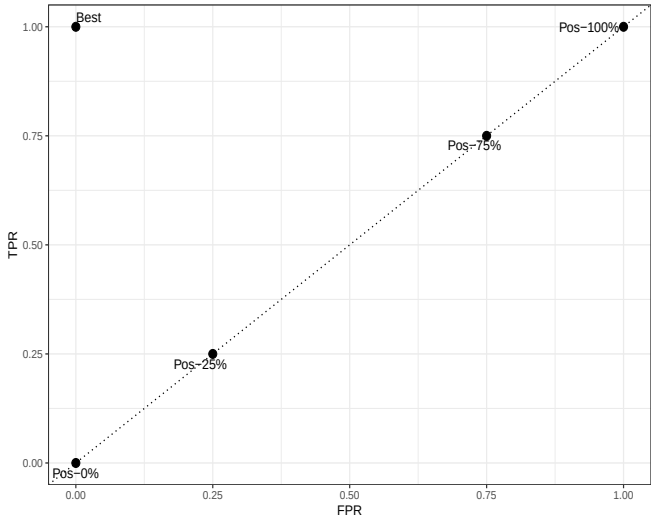
ROC SPACE

- We characterize a classifier by its TPR and FPR values and plot them in a coordinate system
- We could also use 2 different ROC metrics which define a trade-off, like TPR and PPV!



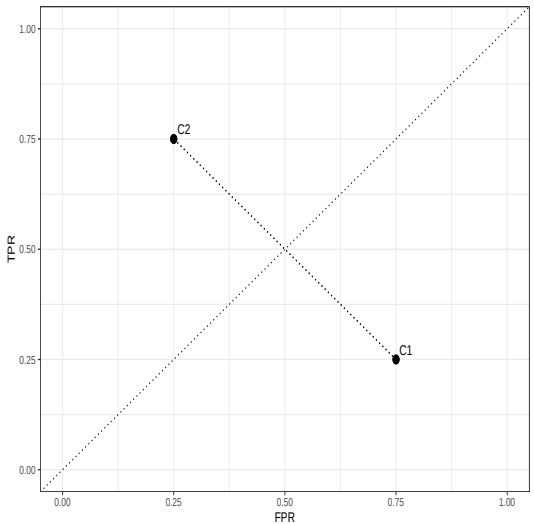
ROC SPACE

- The best classifier lies on the top-left corner
- The diagonal is worst, where classifiers produce random labels (with different proportions). If each positive x will be randomly classified with 25% as "pos", $TPR = 0.25$. If we assign each negative **x** randomly to "pos", $FPR = 0.25$.



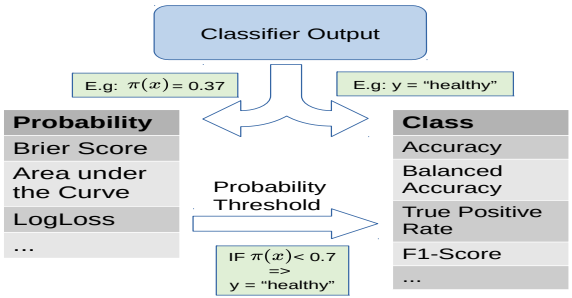
ROC SPACE

In practice, we should never obtain a classifier below the diagonal, as inverting the predicted labels – $0 \to 1$ and $1 \to 0$ – will result in a reflection at the diagonal. Because this inverting results in $TPR2 = 1 - TPR1$ and $FPR2 = 1 - FPR1$



SCORING CLASSIFIERS

- A scoring classifier is a model which outputs scores or probabilities, instead of discrete labels, and nearly all modern classifiers can do that.
- Thresholding flexibly converts measured probabilities to labels. Predict 1 (positive class) if $\hat{f}(\mathbf{x}) > \tau$ else predict 0.
- Normally we could use $\tau = 0.5$ to convert, but for imbalanced or cost-sensitive situations another threshold could be much better.
- After thresholding, any metric defined on labels can be used.



ROC CURVE

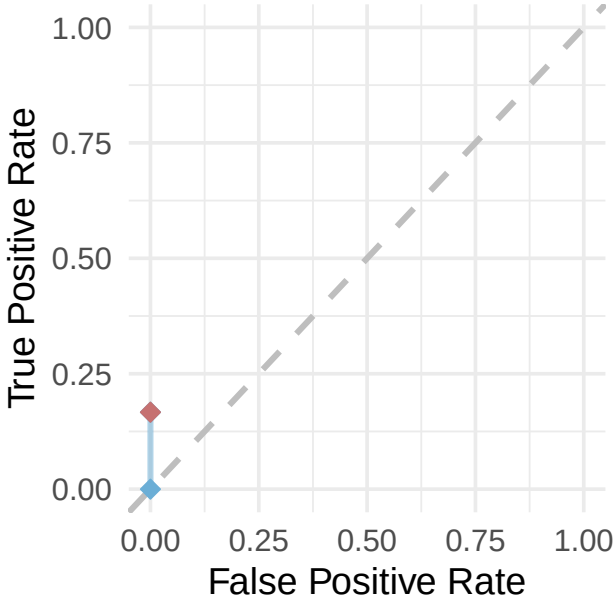
- Are based on thresholding classifiers
- We iterate through all possible threshold, and draw a point in the ROC space (FPR, TPR) for the resulting classifier
- The resulting plot is called an ROC curve
- Small thresholds will very liberally predict class 1, and result in a potentially higher FPR, but also higher TPR
- High thresholds will very conservatively predict class 1, and result in a lower FPR and TPR
- As we have not defined the trade-off between false postives and false negative costs, we cannot easily select the "best" threshold; but a visual inspection of all possible results seems useful

ROC CURVE

- Rank test observations on decreasing score
- Set $\alpha = 1$, so we start in $(0, 0)$; we predict everything as "neg"
- For each observation x (in the decreasing order).
  - Reduce threshold, so prediction for next observation changes
  - If x is "pos", move TPR $1/n_+$ up, as we have one TP more
  - If x is "neg", move FPR $1/n_-$ right, as we have one FP more

ROC CURVE

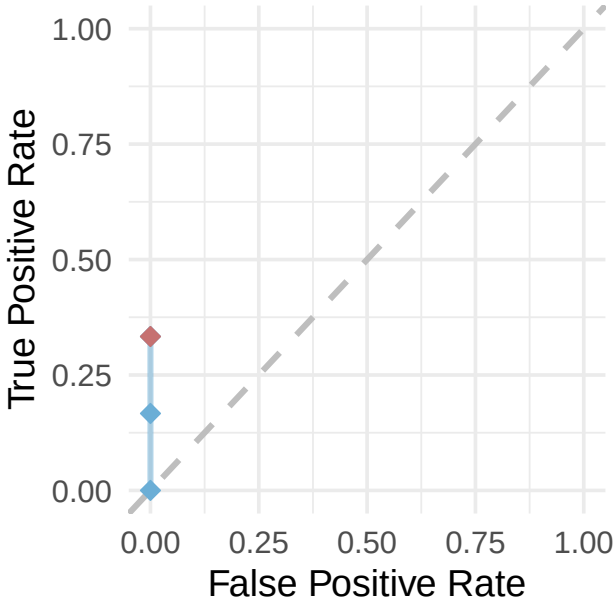
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.9$ yields TPR 0.167 and FPR 0.

ROC CURVE

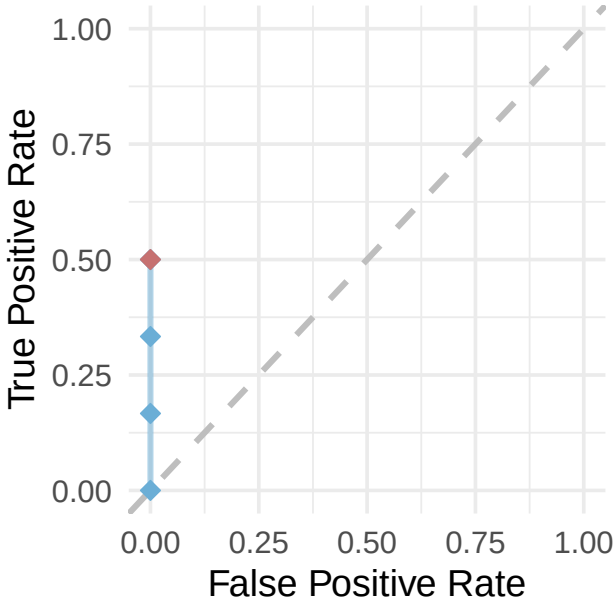
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.85$ yields TPR 0.333 and FPR 0.

ROC CURVE

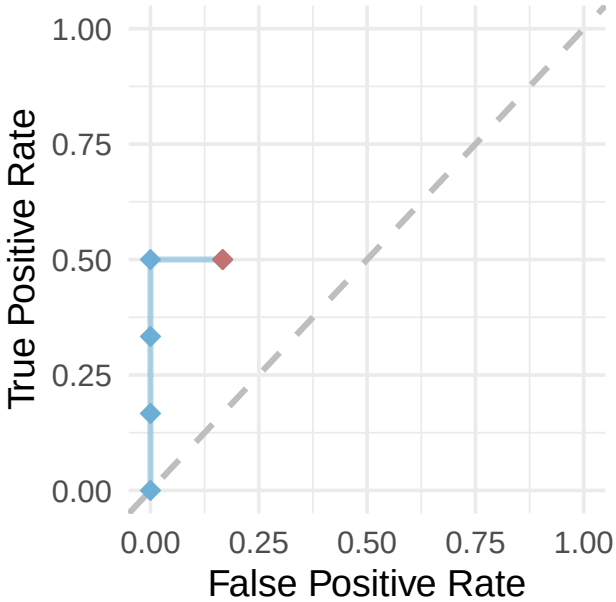
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.66$ yields TPR 0.5 and FPR 0.

ROC CURVE

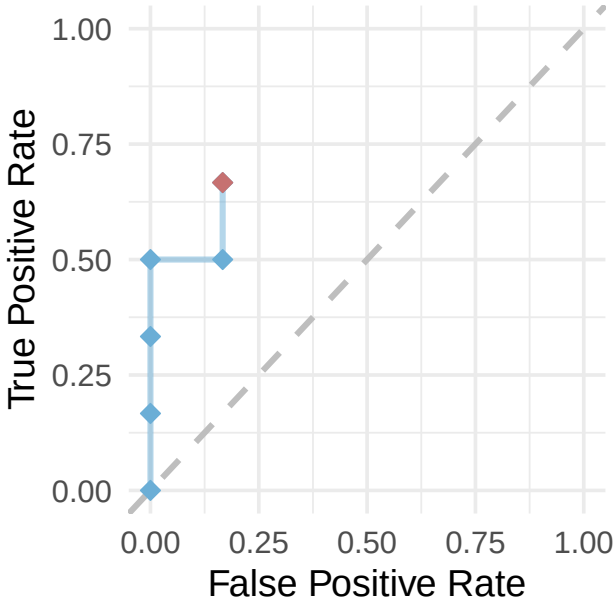
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.6$ yields TPR 0.5 and FPR 0.167.

ROC CURVE

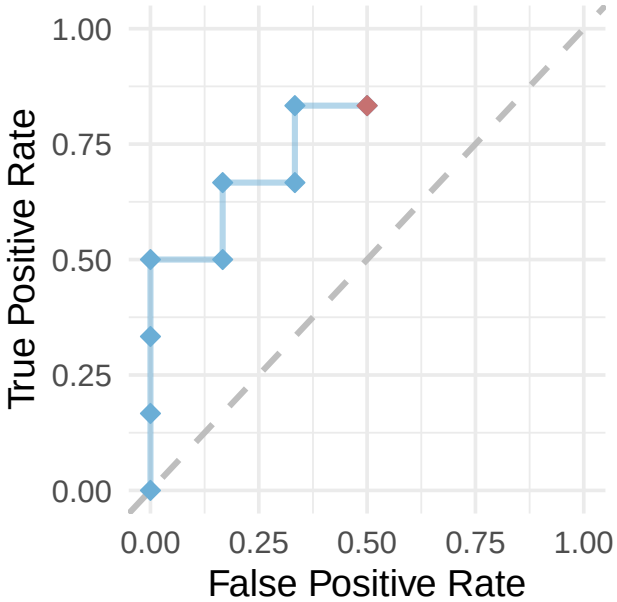
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.55$ yields TPR 0.667 and FPR 0.167.

ROC CURVE

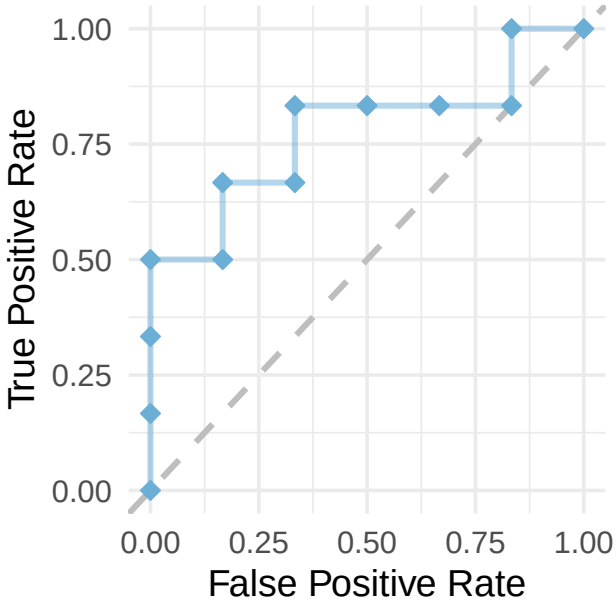
#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



Set threshold $\tau = 0.3$ yields TPR 0.833 and FPR 0.5.

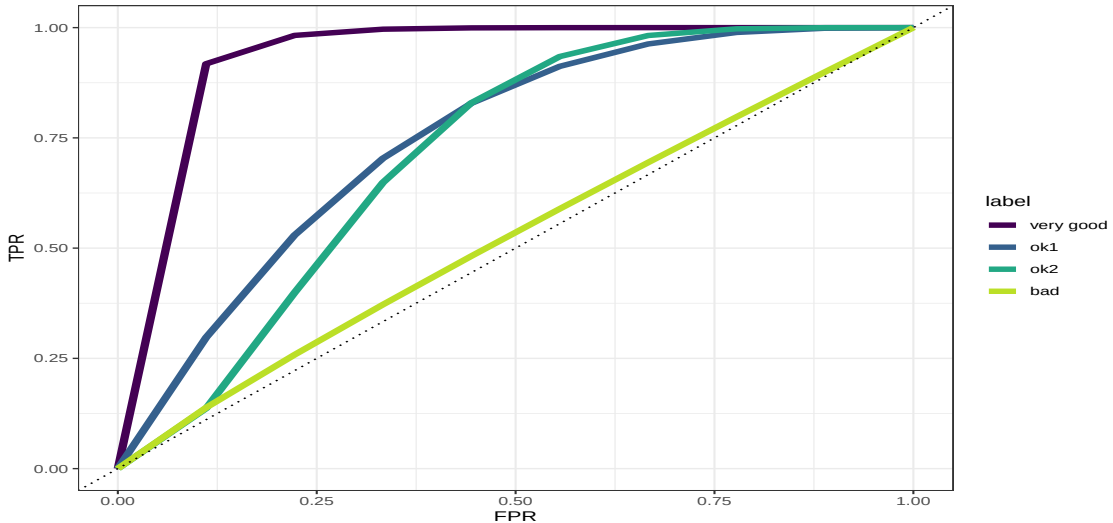
ROC CURVE

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



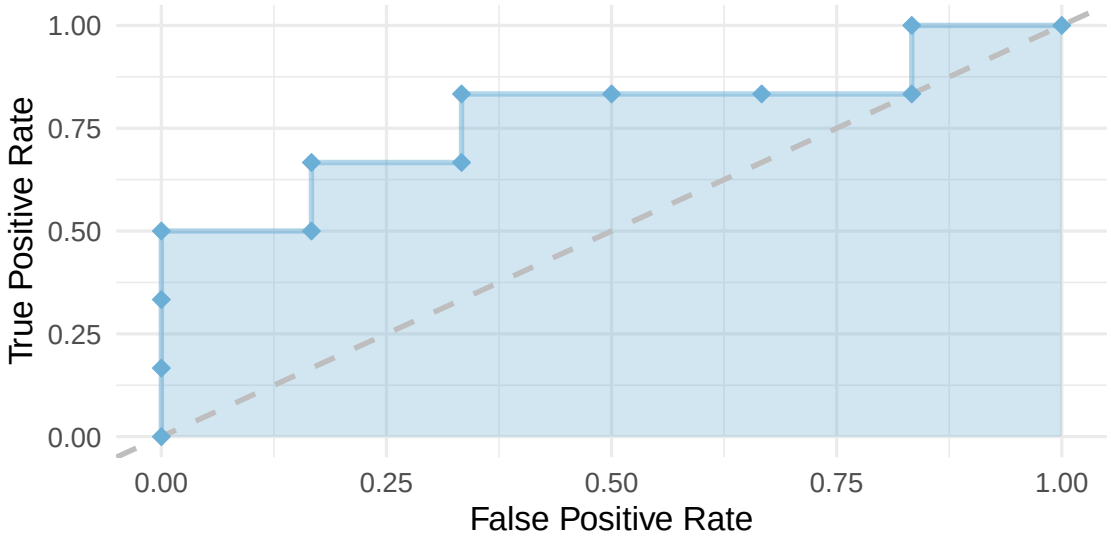
ROC CURVE

- The closer the curve to the top-left corner, the better
- Unfortunately, ROC curves can also cross
- Then, depending on costs and what you want, a different model can be better in different parts of the ROC space



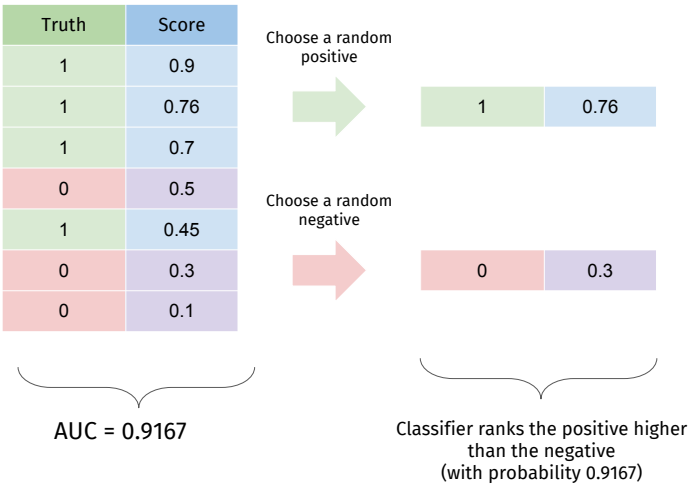
AUC: AREA UNDER ROC CURVE

- The AUC (in [0,1]) is a single metric to evaluate scoring classifiers
- AUC = 1: Perfect classifier
- AUC = 0.5: Randomly ordered
- AUC = 0: Perfect, with inverted labels



AUC: AREA UNDER ROC CURVE

Interpretation: Probability that classifier ranks a random positive higher than a random negative observation



PARTIAL AUC

- Sometimes it can be useful to look at a specific region under the ROC curve $\Rightarrow$ partial AUC (pAUC).
- Let $0 \leq c_1 < c_2 \leq 1$ define a region.
- For example, one could focus on a region with low FPR ($c_1 = 0, c_2 = 0.2$) or a region with high TPR ($c_1 = 0.8, c_2 = 1$):