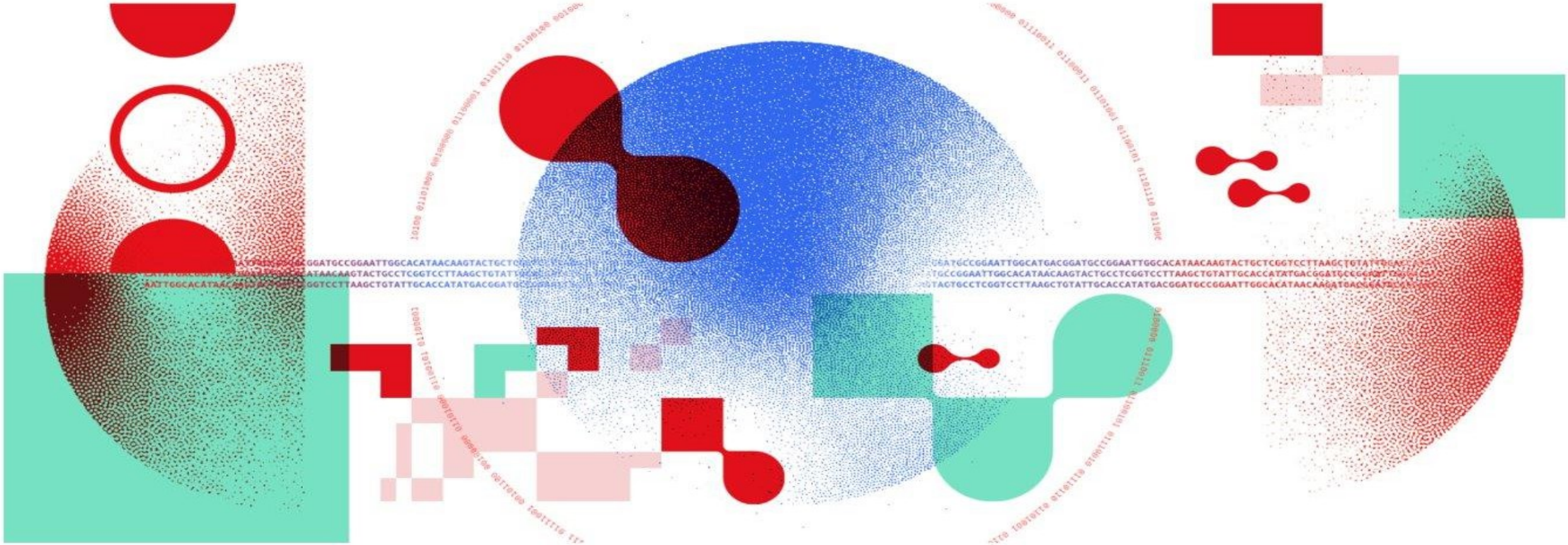


Advanced Statistical Modelling

Lausanne, September 2026

Paolo Angelino and Rachel Marcone

Caret Day

Plan

- Model Performance
- Sensitivity-Specificity
- ROC curve and Area under the ROC curve (AUC)
- Regularization
- k-fold Cross validation and Leave-one-out method (L1O)
- Signatures

Model Performance-General rules

- We have seen so far that if you compare two models that are nested you can use ANOVA between the two models.

- In order to describe the best model (at least so far), there are measures to describe how good the model is.

- You can use the sum of the residuals, the AIC or BIC, or you could split the original data into a test and training set, assess the parameters on the training set and can therefore use the training set to understand how good the model is. This is often used and there are several measurements that can be retrieved using this method.

AIC (Akaike Information Criterion

For a model with k parameters and maximum likelihood L_{max},

$$\text{AIC} = 2k - 2\ln(L_{max})$$

- Interpretation: Lower AIC → Better model (only for relative comparison)
- Penalizes overfitting by adding cost for extra parameters.
- Usage: Compare multiple models and choose the one with the lowest AIC.
- Key Point: AIC does not test absolute model quality — it's for comparing models.

BIC (Bayesian Information Criterion)

Similar to AIC, but includes a stronger penalty for model complexity.

$$\text{BIC} = \ln(n)k \ - 2\ln(L_{max}),$$

where n = number of observations, k = number of parameters, L_{max} = maximum likelihood

- Interpretation: Lower BIC → Better model (relative comparison).
- Penalizes overfitting more strongly than AIC, especially for larger datasets.
- Usage: Often preferred when sample size is large.
- Key Point: Like AIC, BIC is for comparing models, not assessing absolute quality.

RSME (Root Mean Squared Error)

A measure of the differences between predicted and observed values.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i}(y^{i}_{pred} - y^{i}_{true})^{2}}$$

- Interpretation: Lower RMSE indicates better fit; heavily penalizes large errors.
- Sensitive to outliers because errors are squared before averaging.
- Usage: Common in regression and forecasting evaluation.
- Key Point: RMSE is in the same units as the target variable.

MAE (Mean Absolute Error)

The average of the absolute differences between predicted and observed values.

$$\text{MAE} = \sqrt{\frac{1}{n}\sum_{i} | y^{i}_{pred} - y^{i}_{true} |}$$

- Interpretation: Lower MAE indicates better fit; treats all errors equally.
- Less sensitive to outliers than RMSE.
- Usage: Common metric for regression performance evaluation.
- Key Point: MAE is in the same units as the target variable.

Accuracy and Kappa

- Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)
- Does not tell you about if it is better at predicting 1 or 0s and is not telling you either about the balance between the two
- Overall measure of performance

- Kappa = measures the **agreement between predicted and actual class labels**, adjusted for agreement occurring by chance.

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Where:

- p_o = observed accuracy (same as regular accuracy).
- p_e = expected accuracy by random chance.

$$p_e = (p_{A1} \cdot p_{P1}) + (p_{A0} \cdot p_{P0})$$

Where:

- p_{A1}: Proportion of actual class 1
- p_{P1}: Proportion of predicted class 1
- p_{A0}: Proportion of actual class 0
- p_{P0}: Proportion of predicted class 0

Sensitivity vs Specificity

- **Sensitivity** = **true positive rate**, This measures the proportion of individuals with the disease (or value 1 in the dummy category) who are correctly identified as positive by the test.
- **Specificity** = **true negative rate**, is the percentage of persons without the disease (or value 0) who are correctly excluded by the test.

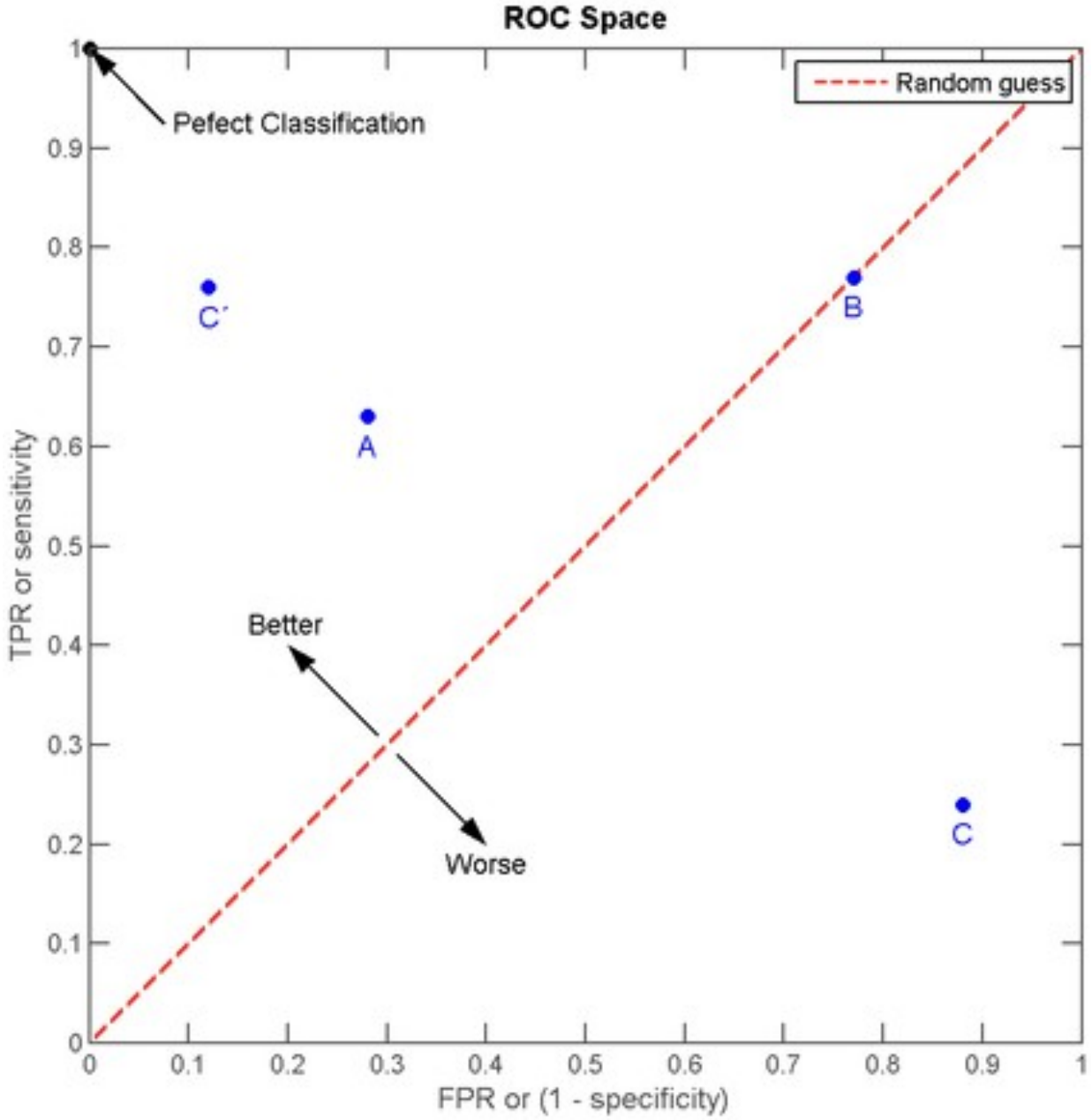
- **High sensitivity**:
- Good at detecting disease, even if it might miss some true negatives (false positives).
- **High specificity:**
- Good at excluding disease, even if it might miss some true positives (false negatives).
- Cancer diagnostic test often have a high **sensitivity** but a lower specificity (because then they get confirmed with a biopsy or secondary test) (specifically for instance with the PSA blood test in prostate cancer)

- Pregnancy test might be more important to have a high specificity because you do not want nonpregnant women to be called pregnant with the pregnancy test even if then you might miss some women that are pregnant, and the test said not pregnant.

ROC curve

- The receiver operating characteristic curve, or ROC curve, is a **graphical plot** that illustrates the performance of a binary classifier model at varying threshold values.
- The ROC curve is the plot of the true positive rate (TPR) against the false positive rate (FPR) at each threshold setting.
- The TPR = sensitivity
- The FPR = 1-specificity (called also probability of false alarm)

ROC



*wikipedia, ROC curve

"I only have my data, how can I make several points for my ROC curve"

- 1. Calculate the probability for all the elements in your test set (predict function in R with type ="response").
- 2. Sort them.
- 3. Now for each value in the response vector you will calculate the number of TP/TN rates and these will be the points in your ROC curve. (Using for example ROCR::prediction and ROCR::performance)
- 4. These values are used to draw the curve.

Example in R using ROCR package

```
> true_labels <- c("P","H","P","H","H")
> predict_proba <- c(0.03,0.1,0.8,0.9,0.95)
> predi <- ROCR::prediction(predict_proba,predict_file)
> perf <- ROCR::performance(predi,x.measure="spec",measure="sens")
> perf@x.values
[[1]]
[1] 1.0000000 0.6666667 0.3333333 0.3333333 0.0000000 0.0000000

> perf@y.values
[[1]]
[1] 0.0 0.0 0.0 0.5 0.5 1.0
```

Example in R using ROCR package

```
> true_labels <- c("P","H","P","H","H")
> predict_proba <- c(0.03,0.1,0.8,0.9,0.95)
> predi <- ROCR::prediction(predict_proba,predict_file)
> perf <- ROCR::performance(predi,x.measure="spec",measure="sens")
> perf@x.values
[[1]]
[1] 1.0000000 0.6666667 0.3333333 0.3333333 0.0000000 0.0000000

> perf@y.values
[[1]]
[1] 0.0 0.0 0.0 0.5 0.5 1.0
```

First we start with the cutoff Infinity. Meaning I have predicted all the samples to be **healthy** : specificity = 1, sensitivity = 0

Then the cutoff will be 0.95 so all my samples will be H except the last one, which is predicted wrongly to be a patient : Specificity = 2/3 healthy are correct, Sensitivity = 0, none of the patient have been identified as patients

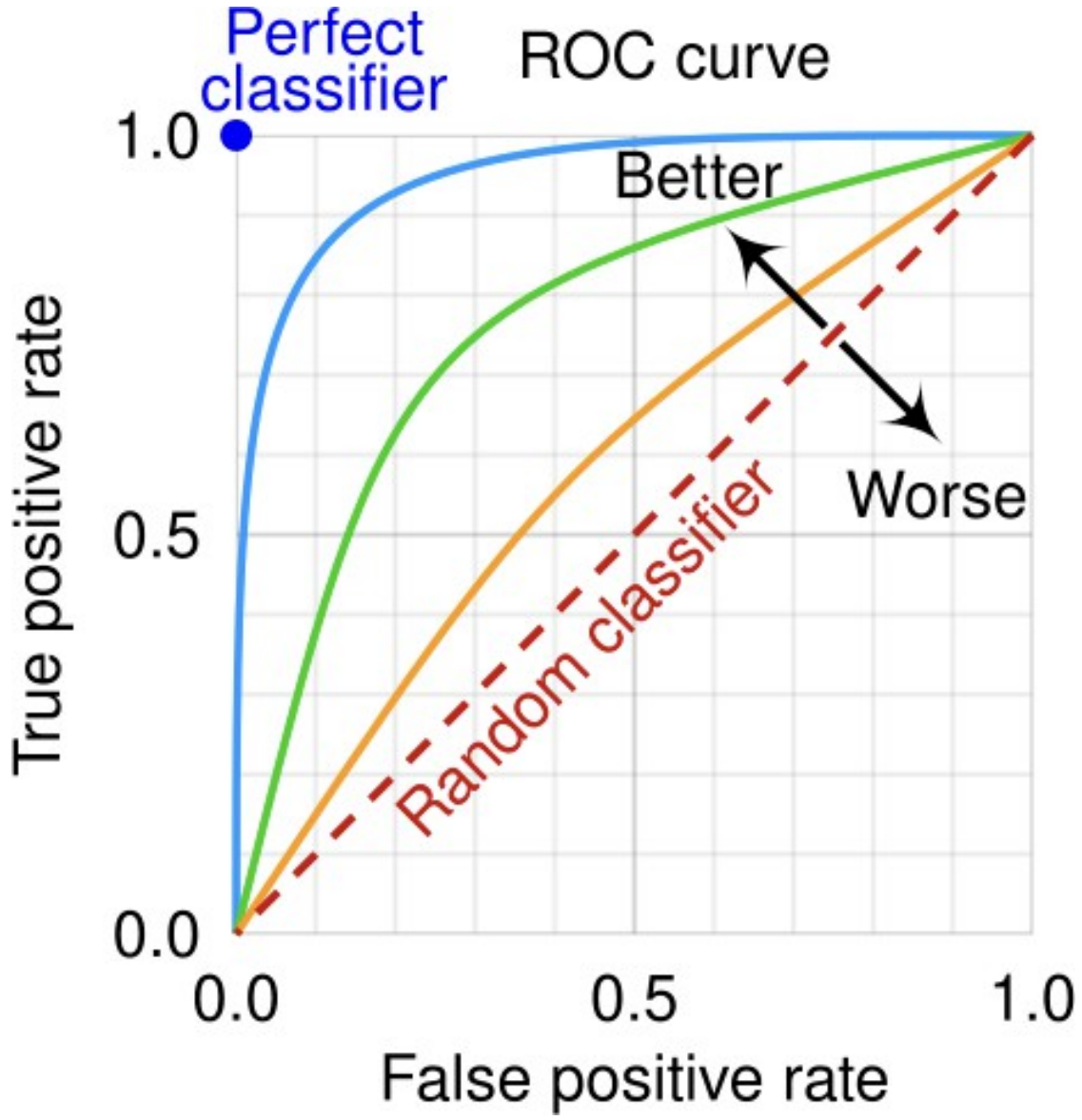
Then cutoff will be 0.9 so all my samples except the last two are considered H, the last two are P : Spec = 1/3 healthy are correct, sensi = 0 none of the P are correct

Then the cutoff will be 0.8, so first two are H last 3 are patients, now my Spec = 1/3 and my sensi = ½

Then cutoff of 0.1, only my first is H all the others are P. The speci= 0, the sensi = ½

Last all the samples are called P and I have a spec= 0,sensi = 1

ROC



*wikipedia, ROC curve

AUC

- Area under the curve or area under the ROC curve.
- A perfect AUC would be of a square 1x1 = 1
- A really random classifier would be giving you a value on the red line, and therefore an AUC of 0.5
- Then, the AUC should always be between 0.5 and 1.

- What happens if your AUC is below 0.5 ? This means you have a higher number of false positives and a lower number of true positives, i.e. the better model would be with inversing all the samples that are called 1 and 0.

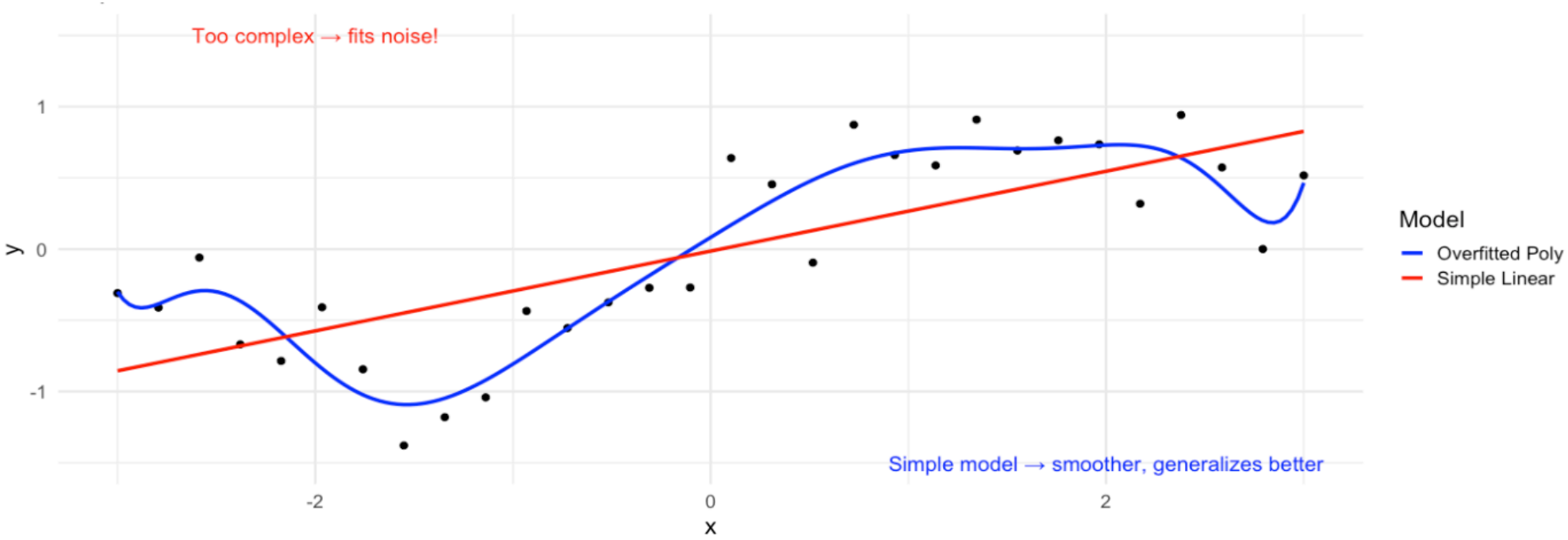
Why AUC should be used with caution

- "A method of comparing the areas under receiver operating characteristic curves derived from the same cases." Hanley and Neil 1983 describes how to compare AUC of ROC curves and therefore performance of predictors on same cases.
- However AUC is a summary of the ROC curve with only one value and therefore you miss all the obtained information.
- AUC is also not able to show overfitting or understand the implied measurements.
- Still 1. It is a measure often used in machine learning
- 2. There are many papers trying to convince why it should not be used anymore

Overfitting

- When: model fits training data too closely
- Captures noise instead of underlying pattern
- High training accuracy but poor generalization
- Occurs when model complexity is too high relative to data
- Mitigation: regularization, cross-validation, or simpler models

Overfitting



Regularization

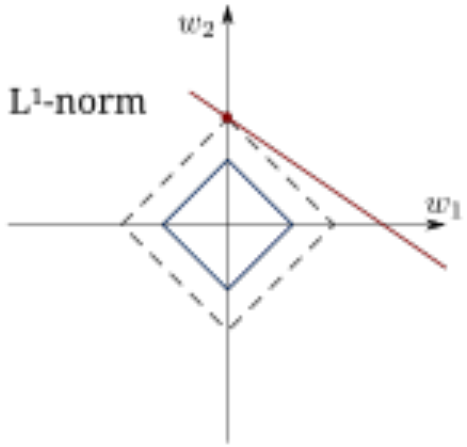
- Regularizations are regressions that optimize a function with a penalty on the coefficients that are used for prediction of the outcome.
- This reduces the variance in the data and reduce overfitting and getting a better balance for the parameters.
- Some regularization will "remove" coefficients that are unnecessary for the regression
- Regularization methods seek to both minimize the sum of squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model

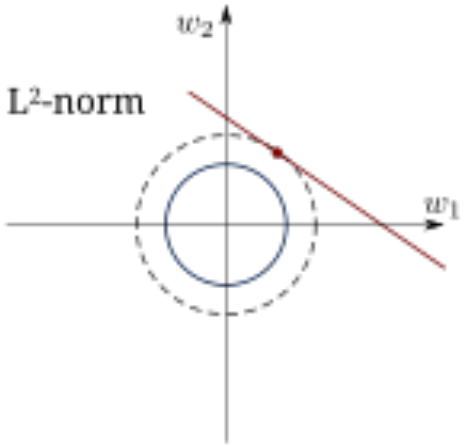
3 popular methods

- Lasso Regression (least absolute shrinkage and selection operator): Ordinary least squares is modified to also minimize the absolute sum of the coefficient (called also L1-regularization)
- Ridge Regression: Ordinary least squares is modified to also minimize the squared sum of the coefficient (called also L2-regularization)
- Elastic Net Regression : A combination of both of the above.

- Useful when : you have many parameters that you want to use to predict the model, when you have colinearity, or when you want to be more sure of the prediction you do.

Formulas





Lasso

$$\hat{w} = \arg\min_{w} ||Y - Xw||^2 + \lambda_1 ||w||_1$$

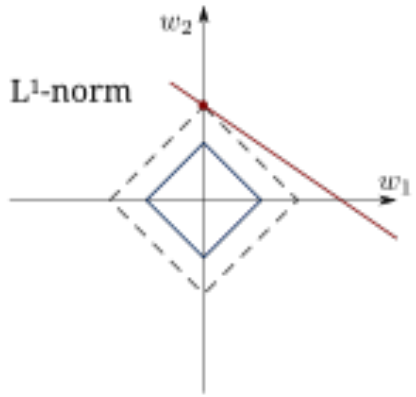
Ridge Regression

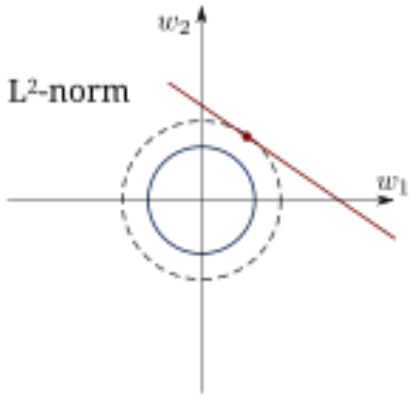
$$\hat{w} = \arg\min_{w} ||Y - Xw||^2 + \lambda_2 ||w||_2^2$$

Elastic net

$$\hat{w} = \arg\min_{w} ||Y - Xw||^2 + \lambda_1 ||w||_1 + \lambda_2 ||w||_2^2$$

Formulas





Elastic net

Often written as

$$\hat{w} = \arg\min_{w} ||Y - Xw||^2 + \lambda \left(\alpha||w||_1 + (1-\alpha)||w||_2^2\right)$$

➔ Alpha = 1 is LASSO

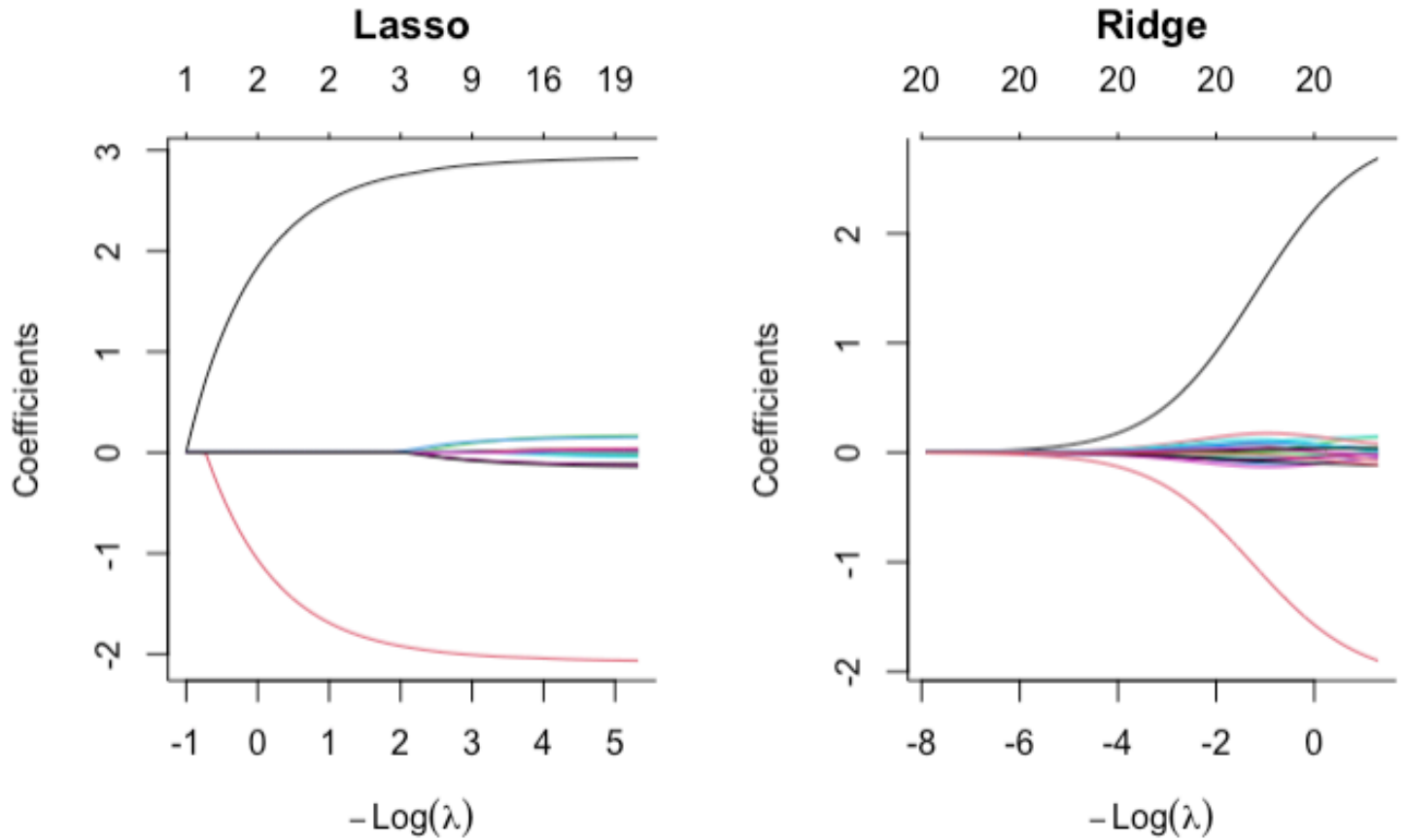
➔ Alpha = 0 is Ridge

Regularization

- The lambda parameters (must be tuned by the user. They are what me call **hyperparameters**.

- To make a good choice, one typically uses:
    - some criterion (AIC, BIC,…)
    - cross-validation

Regularization



Regularization

- Once tuned, Lambda gives us an information about how much parameters needed tuning and how correlated the predictors were .
- If Lambda is high => lot of correlation => might be wise to do a 2by2 correlation plot to understand which information are correlated
- If Lambda is low => almost like doing a standard regression.

Generate a signature

- After having found a list of N favourite genes that should be able to discriminate your two condition (for example Healthy vs Disease) you can already try to see how they perform in univariate modelling.
- But usually, the idea is to build a signature using part of those N genes.
- We can then put these N genes into our new model and with elastic net regression for example find which genes are not necessary.

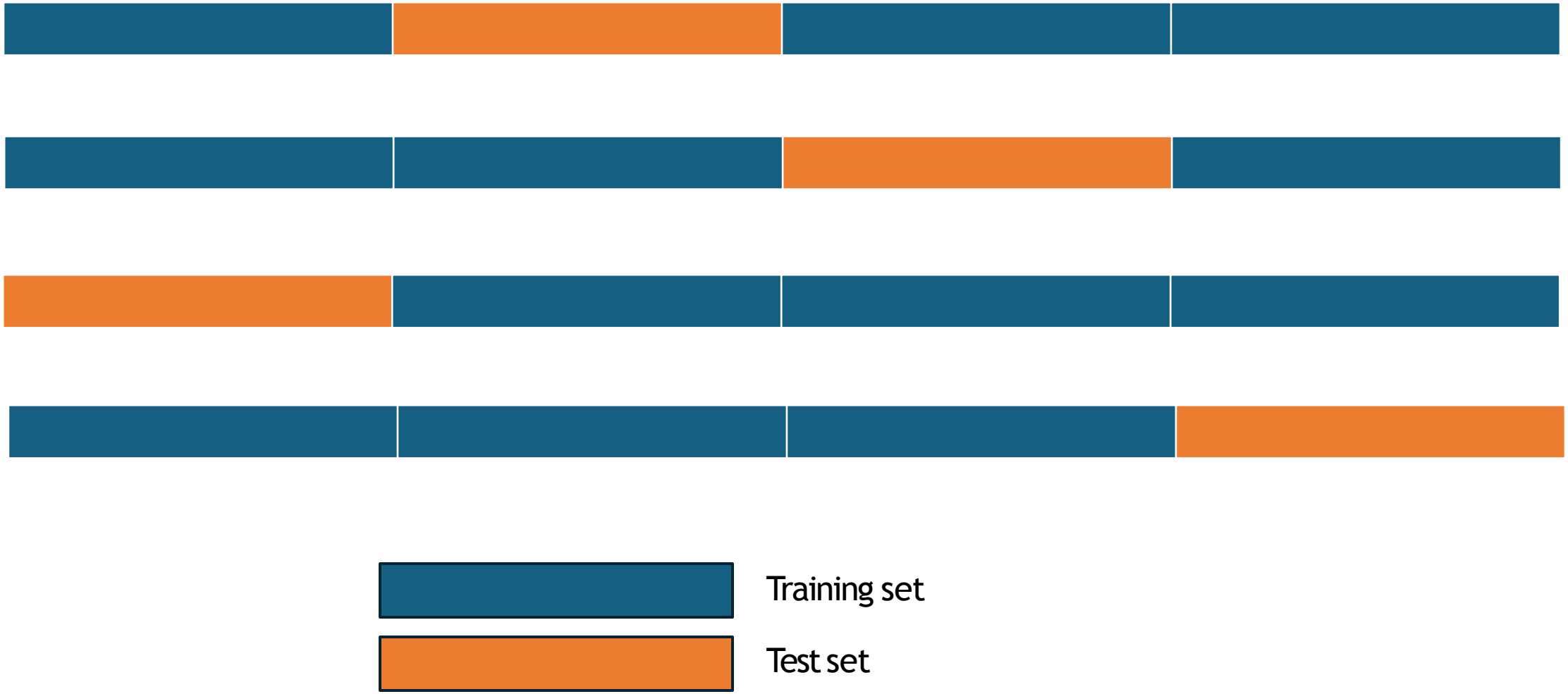
Cross-validation

- … or how to tune hyperparameters.
- … or how to quantify how accurate the model is.
- … or how to know how the model varies according to the points that are included
- … or how to select a model
- … or how to exclude variables
- … or how to show overfitting
- … or how to assess how good the model is

Cross-validation explained

- Separates the dataset into a training and a test set
- Train the model (i.e. find the coefficients) on the training set.
- Test the model (i.e. find the outcome) for the test set and see how accurate the model is

K-Fold cross validation with N repetitions



Check out Josh Starmer's channel

- https://www.youtube.com/watch?v=Q81RR3yKn30 (Ridge Regression)
- https://www.youtube.com/watch?v=NGf0voTMlcs (Lasso Regression)
- https://www.youtube.com/watch?v=1dKRdX9bfIoCt=208s (Elastic net Regression)
- https://www.youtube.com/watch?v=fSytzGwwBVw (cross validation)

- https://www.youtube.com/watch?v=iJE2fZcNPlACt=149s (lecture comparing the 3 first)

CV repeated in R

```
trC.lm <- trainControl(                  #defines the CV procedure
                method = "repeatedcv",   #multiple CV
                number = 5,              #5-fold CV
                repeats = 10)            #repeats the cross validation 10 times

#fits the linear model with CV defined above
model.lm <- train(bmd ~ age + sex + bmi,
                data = bmd.data,
                method = "lm",
                trControl = trC.lm)
```

CV concretely in the case of (G)LMs

1. It is computing a model based on each of the folds.
2. It then tests the test set with the model and parameters estimated in point 1.
3. It averages the measures of the model through all the folds and this gives an estimation of how good the model is. For LM for instance this would be RSME or R^2 or MAE
4. It then re-estimates the parameters based on the full dataset, which can be obtained using model.lm$finalModel

CV in the case of binomial fits

- The measure in the case of binomial fits is ROC curves, specificity and sensitivity. Either one of those could be of higher importance.

```
trC <- trainControl(
            method = "cv",     #just 1 CV
            number = 10,       #10-fold CV
            classProbs = TRUE,
            summaryFunction = twoClassSummary,
            savePred =TRUE)  #to be used in the confusion matrix

model.LR <- train(fract ~ age + sex + bmi + bmd ,
                data = bmd.data, method = "glm",
                family="binomial",
                trControl = trC,
                metric = "ROC")



# Model Summary
model.LR
```

The model.LR$results

- With model.LR$results we obtain the *average* ROC, average sensitivity and average specificity measured on the test set of each folds as well as the standard deviation of them.

```
> model.LR$results
  parameter       ROC      Sens      Spec      ROCSD     SensSD     SpecSD
1      none 0.6126738 0.9793657 0.0967619 0.06596677 0.02078783 0.06788039
```

KNN-method

Method of K-nearest-neighbours

- KNN predicts the label of a new point based on its nearest neighbors

- For example k=5, it would look for the 5 closest neighbours in the data and do an averaging for regression or a most-common result for categorical values.

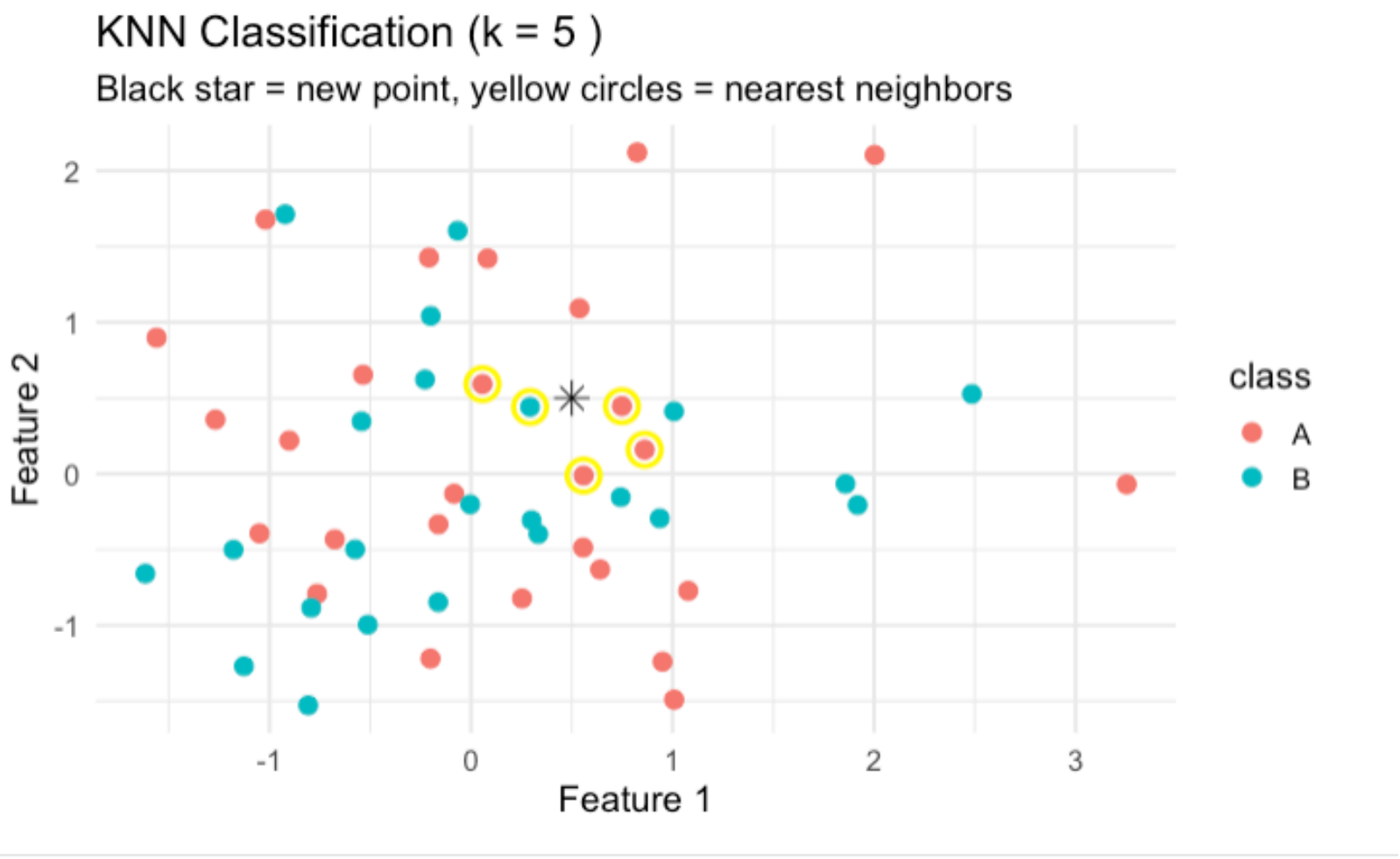
KNN-method

Method of K-nearest-neighbours

- KNN predicts the label of a new point based on its nearest neighbors

- For example k=5, it would look for the 5 closest neighbours in the data and do an averaging for regression or a most-common result for categorical values.

KNN-method


KNN Classification (k = 5)
Black star = new point, yellow circles = nearest neighbors
Feature 2
Feature 1
2
1
0
-1
-1
0
1
2
3
class
A
B

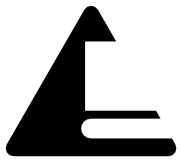
KNN-method

Method of K-nearest-neighbours

- Non-parametric, instance-based learning method
- Used for classification and regression
- Key hyperparameter: number of neighbors (k)
- Sensitive to feature scaling and irrelevant features
- Simple, intuitive, and good for small datasets

Fruit classification

- Suppose you want to classify a new fruit based on weight and color:
- Your training data has labeled fruits: apples, oranges, bananas
- For a new fruit, KNN finds the 3 (k=3) closest fruits according to the model done with weight and color in the training data
- If 2 are apples and 1 is an orange, the new fruit is classified as an **apple**

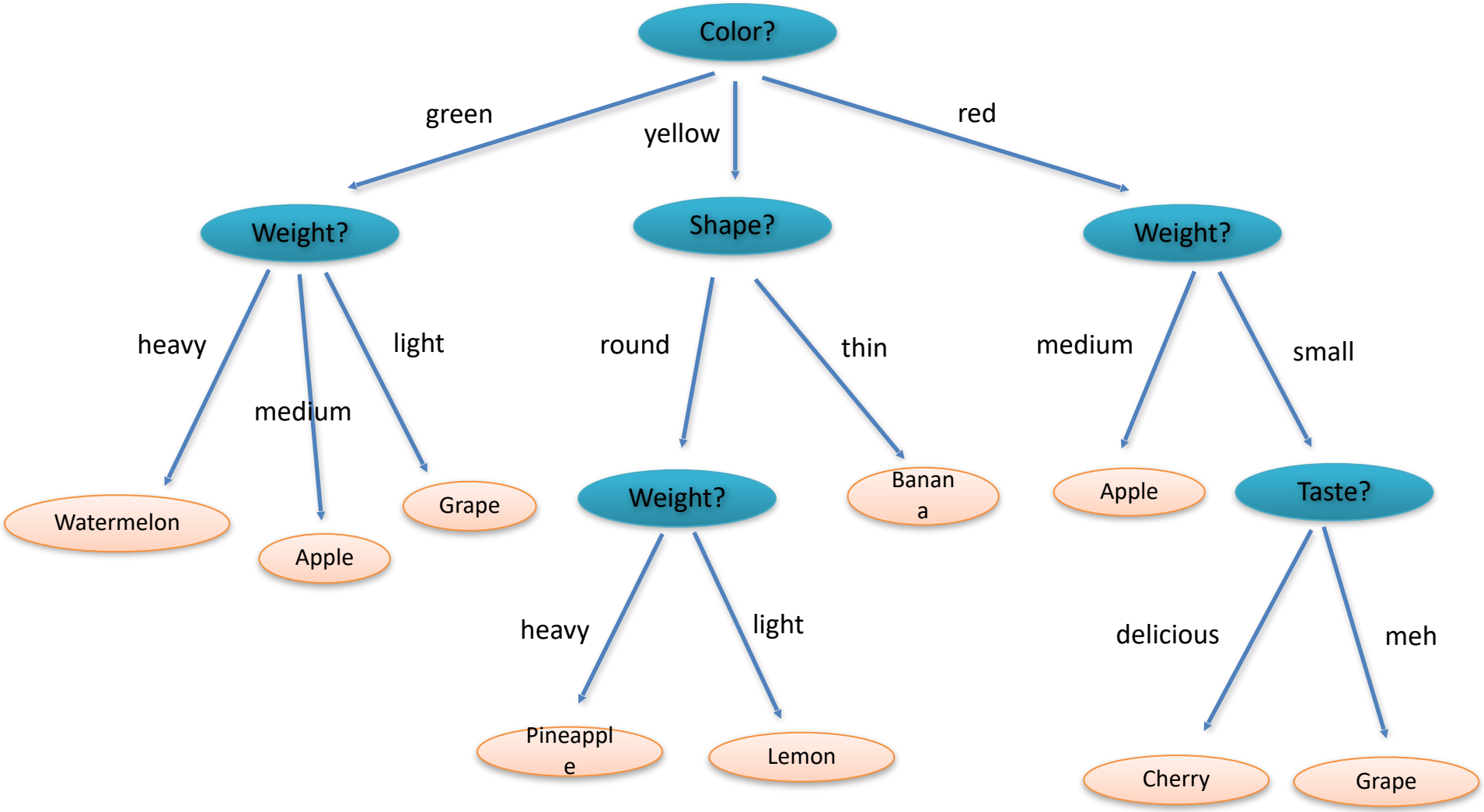


KNN is very dependent on the number K and very dependent on the data. To check the stability of the result use CV

Decision Tree

- A supervised learning algorithm used for classification and regression.
- Models decisions and their possible consequences as a tree structure.
- Splits data into subsets based on feature values.
- Each internal node represents a decision on a feature.
- Each leaf node represents an output or prediction.

Decision Tree



How Decision Trees Work

1. Start with the entire dataset at the root node.

2. Choose the best feature to split the data based on a criterion (e.g., Gini Impurity, Entropy, MSE).

3. Split the data into branches based on the feature's values.

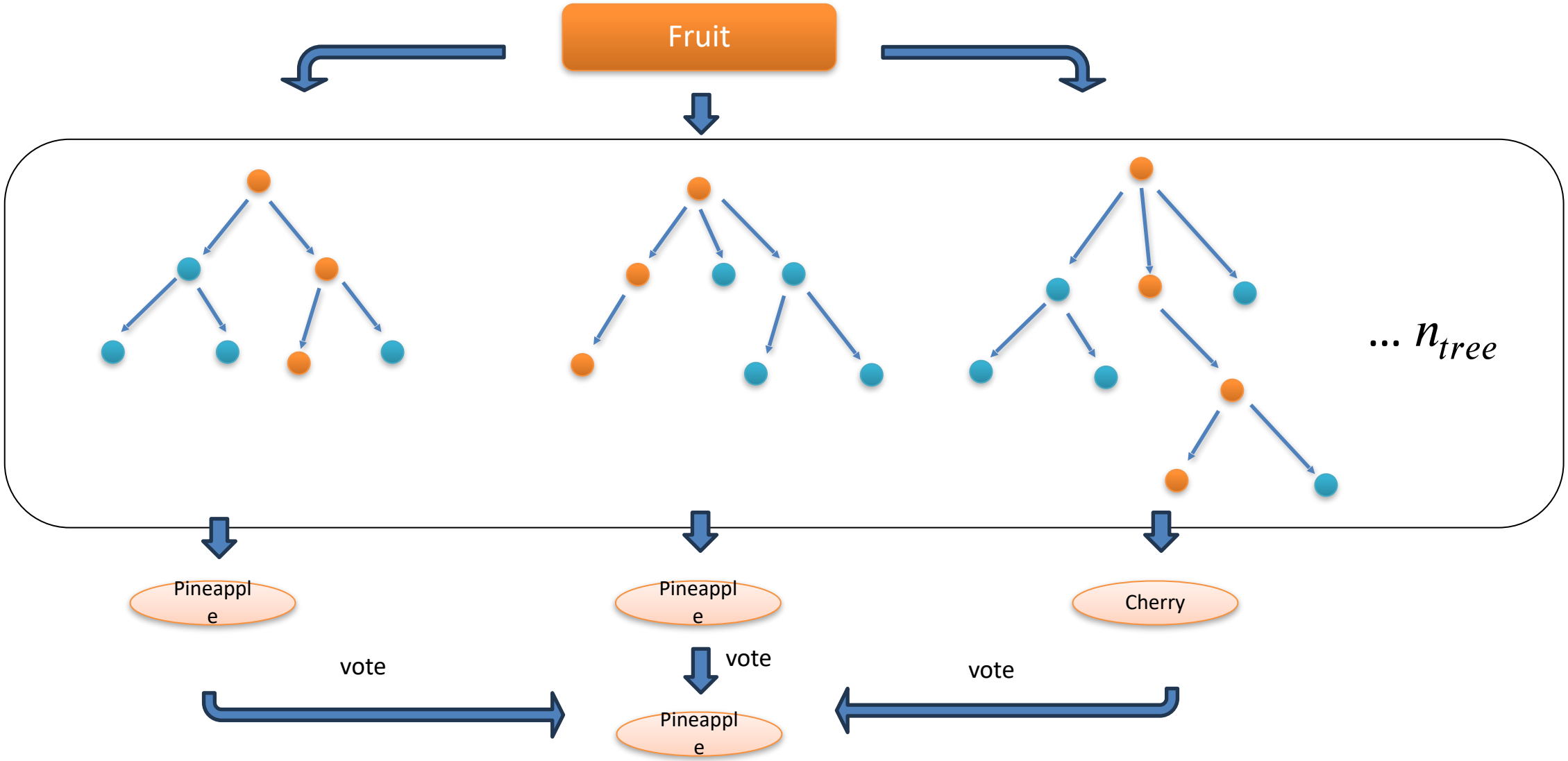
4. Repeat the process recursively for each branch until some stopping criteria are met (e.g., max depth, min samples).

5. Assign the most common class (classification) or average value (regression) to each leaf node.

Random Forest

- A machine learning algorithm that combines multiple decision trees.
- Works for both classification and regression tasks.
- Each tree is trained on a random subset of the data and features.
- The structure of each tree is generated (tree growing) by random sampling of features at each node
- The final prediction is made by averaging (regression) or voting (classification).

Random Forest



How Random Forests Work

1. Bootstrapping: Randomly sample data with replacement to create subsets.

2. Feature Randomness: At each split, choose a random subset of features.

3. Build multiple decision trees from these subsets.

4. Aggregate results from all trees for the final output.

Random Forests parameters

- n_tree – number of trees in the forest
- max_depth – maximum depth of each tree
- min_samples_split – minimum samples to split a node
- min_samples_leaf – minimum samples per leaf node
- max_features – number of features to consider at each split
- criterion – measure for split quality (e.g., gini, entropy, MSE)

Random Forests: Pros and Cons

✅ Pros:

Handles high-dimensional data well
Reduces overfitting compared to single trees
Works for both regression and classification
Robust to noise and outliers
Provides feature importance estimates

⚠️ Cons:

Less interpretable than a single tree
Can be computationally expensive with many trees
Larger memory usage
Slower predictions compared to simple models