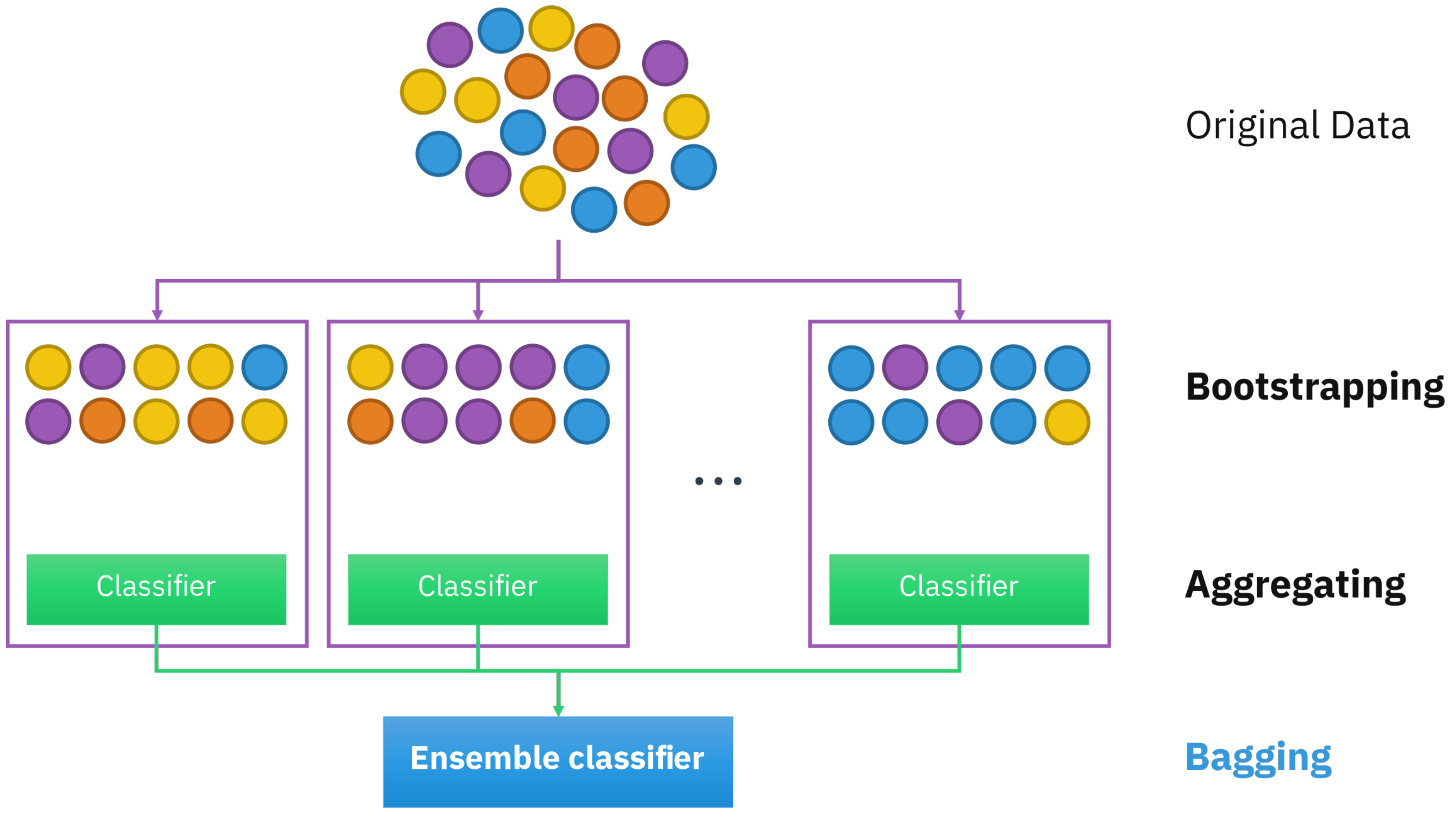
# Day 4

## From Trees to Forests

# Bagging: Random Forests



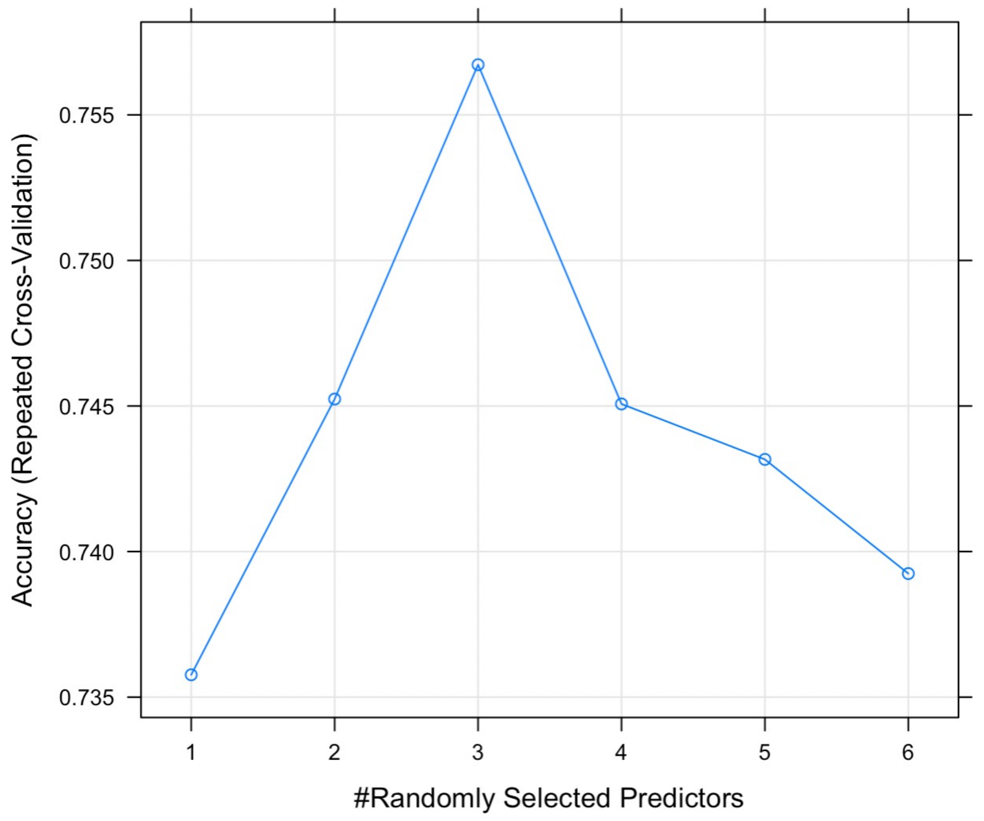
By Sirakorn - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=85888769

## Parameters of the model

- The ***mtry*** parameter controls how many of the input features a decision tree has available to consider at any given point in time.
- Since different sets of features will be available to different decision trees at different points, it will be (nearly) impossible for all of your trees to look exactly the same.
- Predictions are then obtained by taking the average over posterior probabilities or taking the "majority vote"

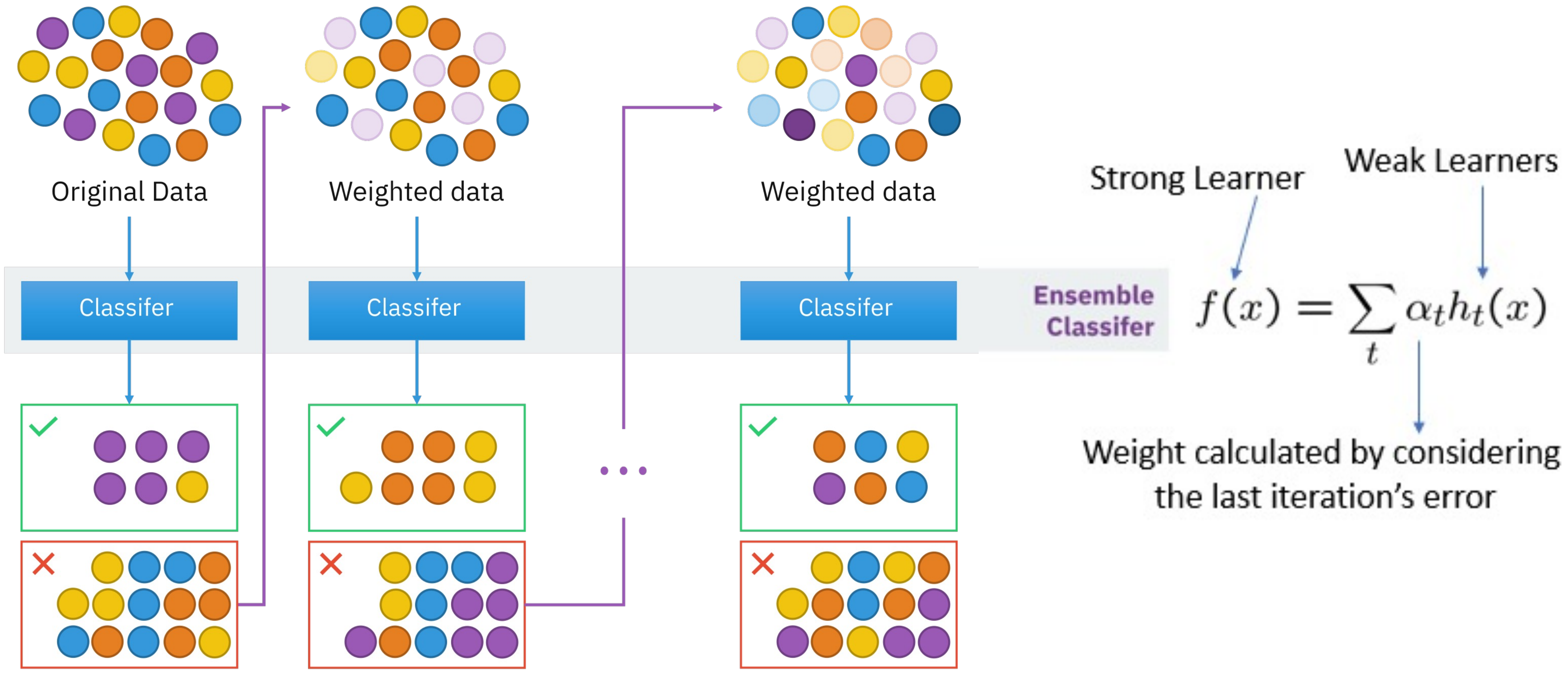
# Setting up a search grid

```
tunegrid <- expand.grid(
  mtry = c(1:6)
)


trcontrol <- trainControl(method = "repeatedcv",
                          number = 5,
                          repeats = 1,
                          search = "grid")

rf_train = caret::train(labels~ .,
                        data=pp_train,
                        trControl = trcontrol,
                        tuneGrid = tunegrid,
                        method = 'parRF')
```



## Boosted trees

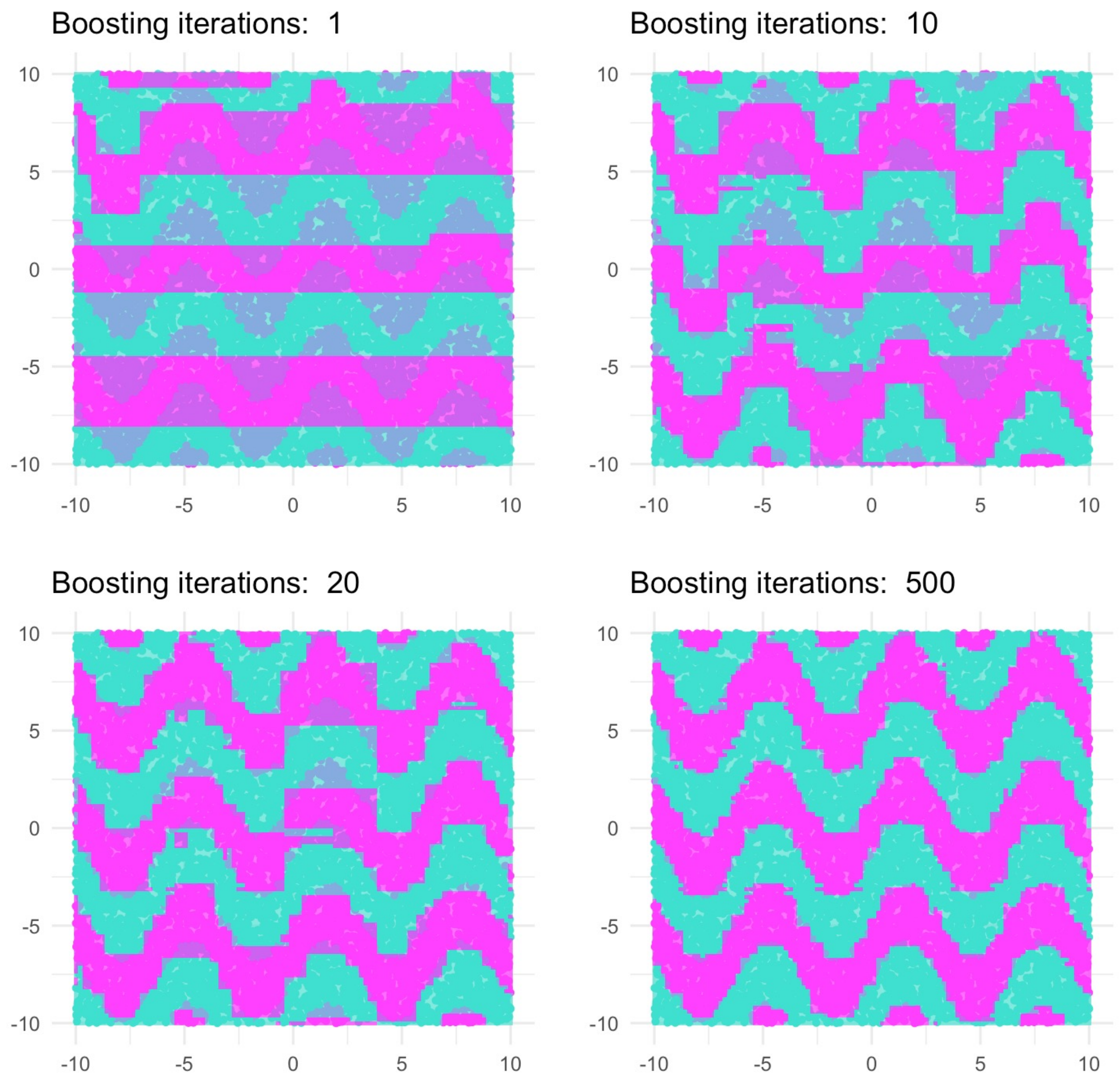
## Boosting



$$f(x) = \sum_t \alpha_t h_t(x)$$

By Sirakorn - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=85888769

# Number of boosting iterations



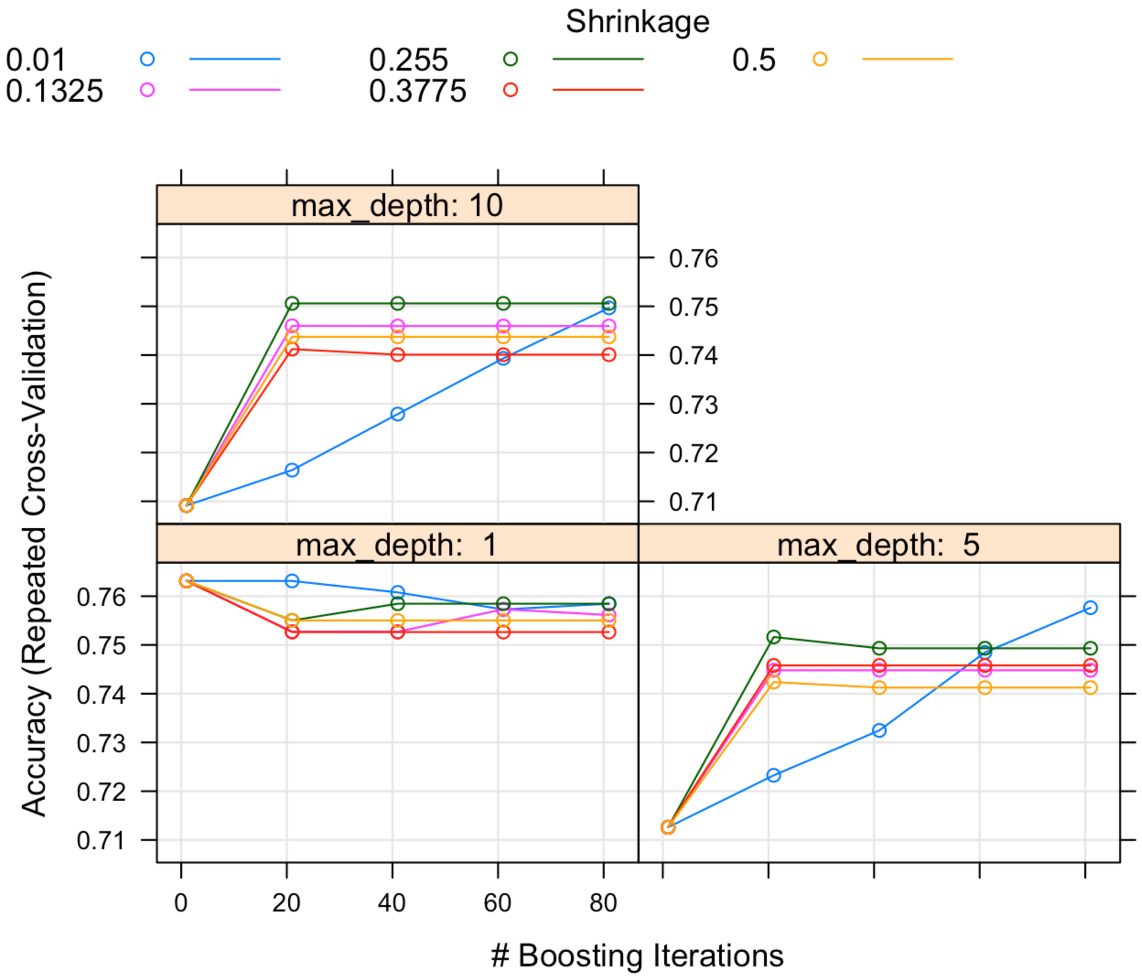
# XGBoost - tunegrid

```
tunegrid <- expand.grid(nrounds = seq(1,100,by=20),
                        max_depth = c(1,5,10),
                        eta = seq(0.01,0.5,length.out = 5),
                        gamma = 1,
                        colsample_bytree = 1,
                        min_child_weight = 1,
                        subsample = 1)
```

```
trcontrol <- trainControl(method = "repeatedcv",
                          number = 5,
                          repeats = 5,
                          search = "random",
                          savePredictions = T)
```

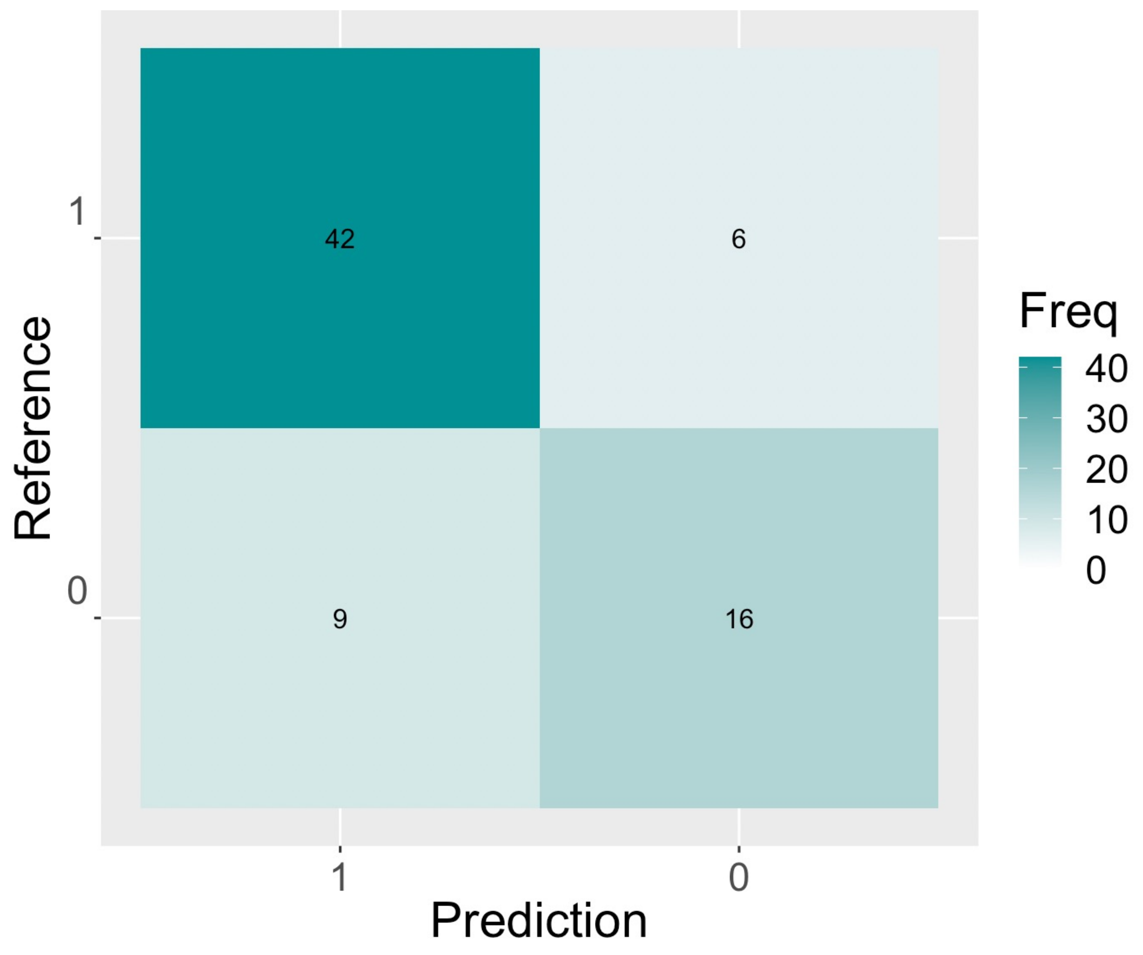
# XGBoost – hyperparameter tuning

```
xgb_train = train(labels~ .,
             data=pp_train,
             trControl = trcontrol,
             tuneGrid = tunegrid,
             method = "xgbTree")


plot(xgb_train)
```

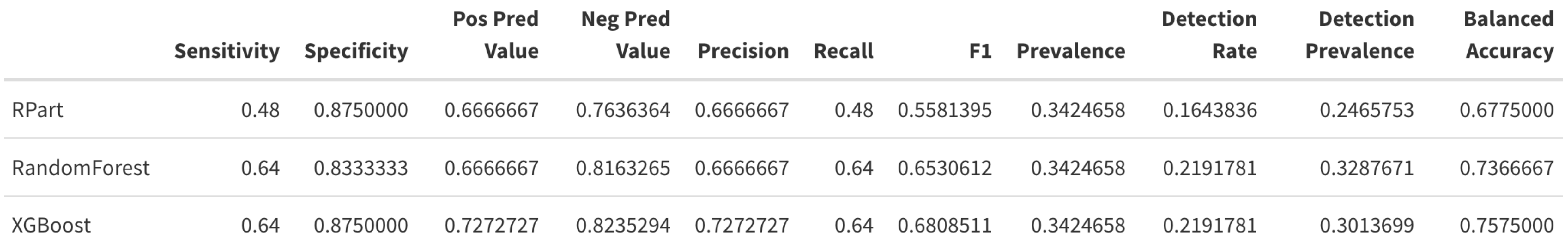


# Confusion Matrix

| | XGBoost |
|---|---|
| Sensitivity | 0.6400000 |
| Specificity | 0.8750000 |
| Pos Pred Value | 0.7272727 |
| Neg Pred Value | 0.8235294 |
| Precision | 0.7272727 |
| Recall | 0.6400000 |
| F1 | 0.6808511 |
| Prevalence | 0.3424658 |
| Detection Rate | 0.2191781 |
| Detection Prevalence | 0.3013699 |
| Balanced Accuracy | 0.7575000 |



# Summary of Models



| | Sensitivity | Specificity | Pos Pred Value | Neg Pred Value | Precision | Recall | F1 | Prevalence | Detection Rate | Detection Prevalence | Balanced Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RPart | 0.48 | 0.8750000 | 0.6666667 | 0.7636364 | 0.6666667 | 0.48 | 0.5581395 | 0.3424658 | 0.1643836 | 0.2465753 | 0.6775000 |
| RandomForest | 0.64 | 0.8333333 | 0.6666667 | 0.8163265 | 0.6666667 | 0.64 | 0.6530612 | 0.3424658 | 0.2191781 | 0.3287671 | 0.7366667 |
| XGBoost | 0.64 | 0.8750000 | 0.7272727 | 0.8235294 | 0.7272727 | 0.64 | 0.6808511 | 0.3424658 | 0.2191781 | 0.3013699 | 0.7575000 |

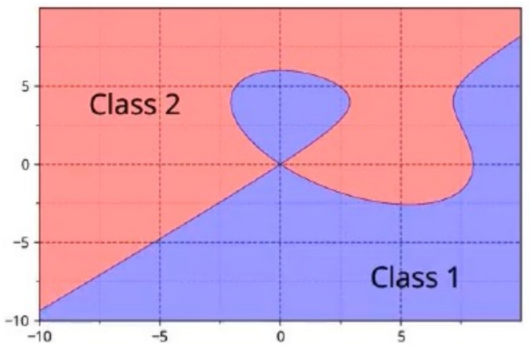
•**Sensitivity or recall** (true positive rate) is the probability of a positive test result, conditioned on the individual truly being positive.
•**Specificity** (true negative rate) is the probability of a negative test result, conditioned on the individual truly being negative.

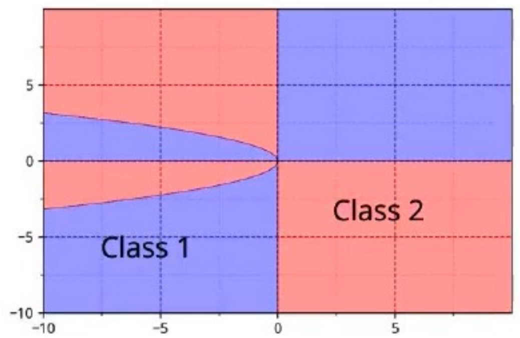
•The **positive and negative predictive values** are the proportions of positive and negative results that are true positive and true negative results, respectively. Positive predictive value is also called **Precision**.
•The **F1** score is defined as the harmonic mean of precision and recall.

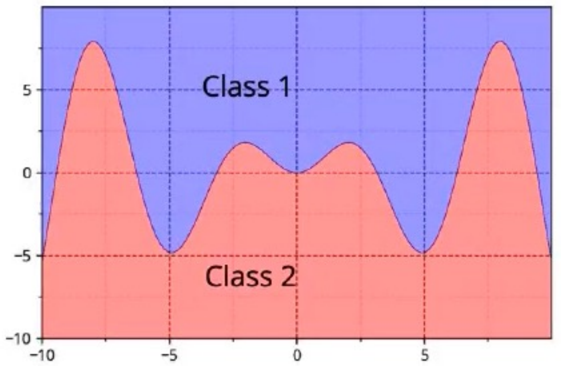
•**Detection rate** is the proportion of the whole sample where the events were detected correctly.
•**Detection prevalence** is the proportion of the whole sample that were classified as the focal category (= “not surviving” in our case).
•**Balanced Accuracy** is the average of specificity and sensitivity.

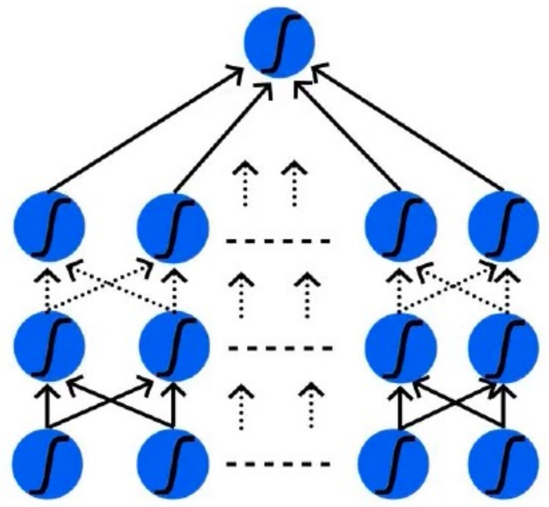
## Neural Networks

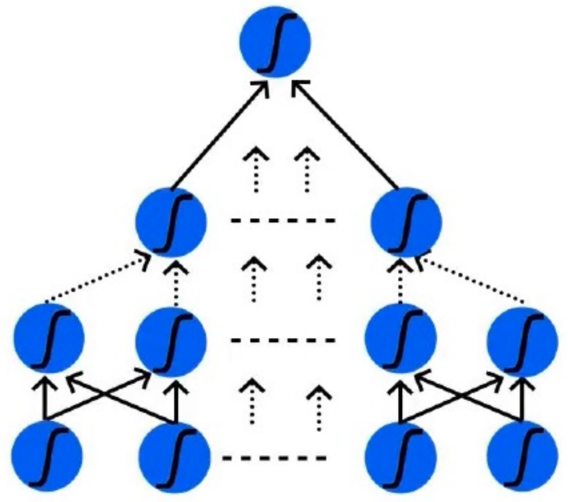
# Neural Networks: universal approximation theorem

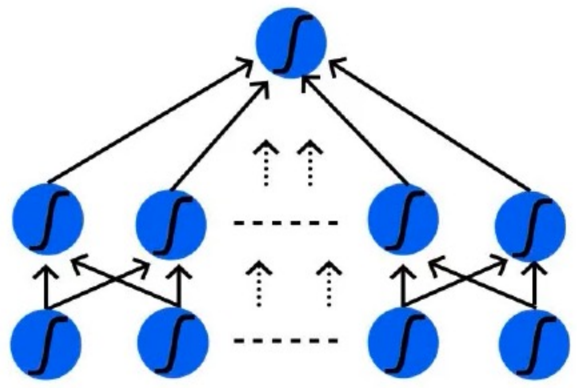






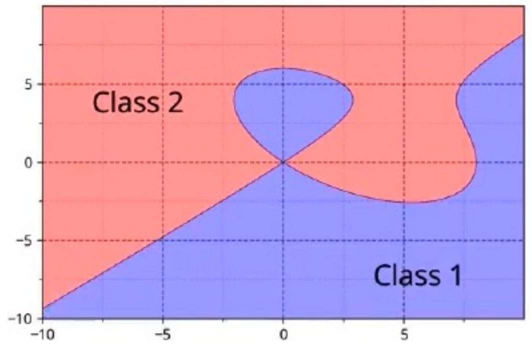


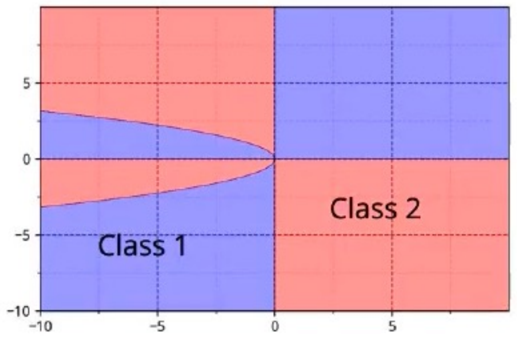


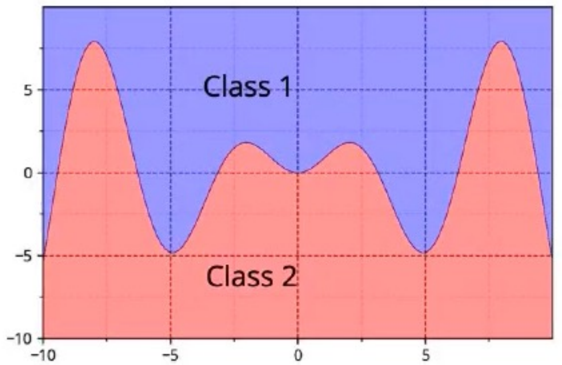


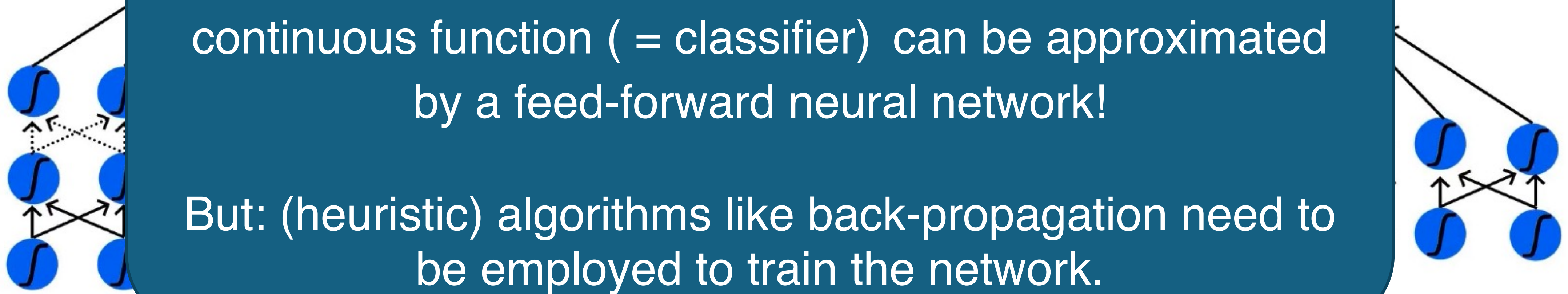
https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6

# Neural Networks: universal approximation theorem









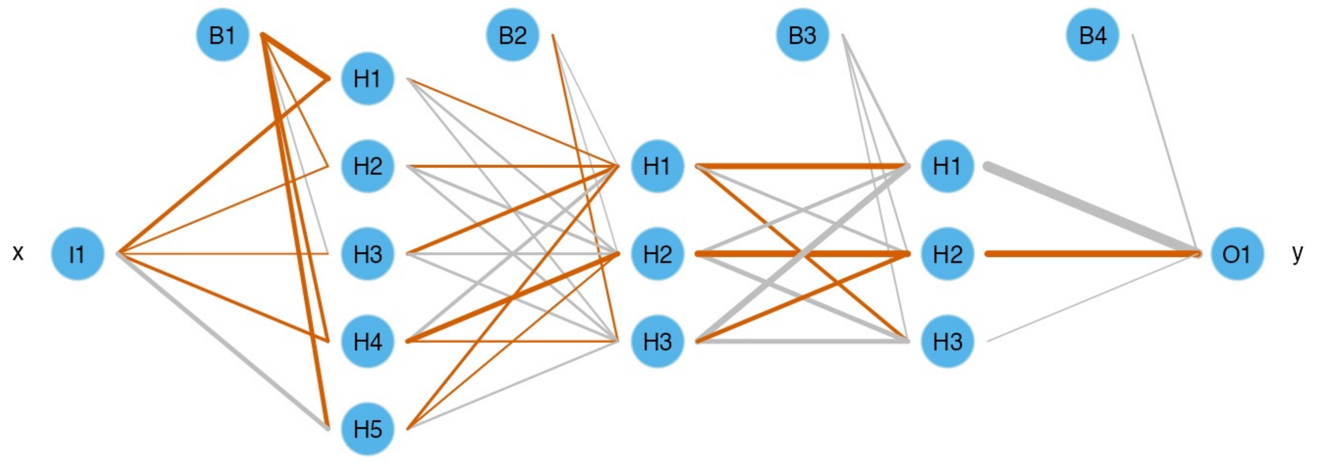
The *universal approximation theorem* states that any continuous function ( = classifier) can be approximated by a feed-forward neural network!

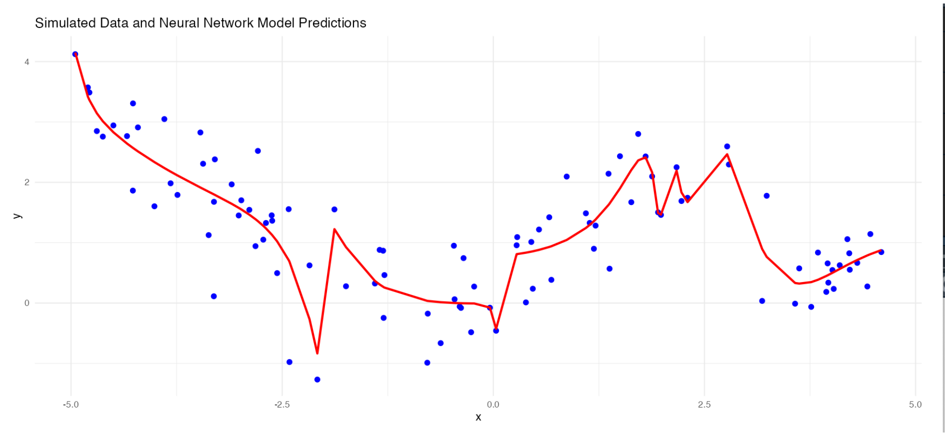
But: (heuristic) algorithms like back-propagation need to be employed to train the network.

https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6

## Network Architecture

- **Input Layer**: Receives the initial data.
- **Hidden Layers**: Intermediate layers performing computations/feature transformations.
- **Output Layer**: Produces the final prediction/output.





## Hyperparameters (list not exhaustive)

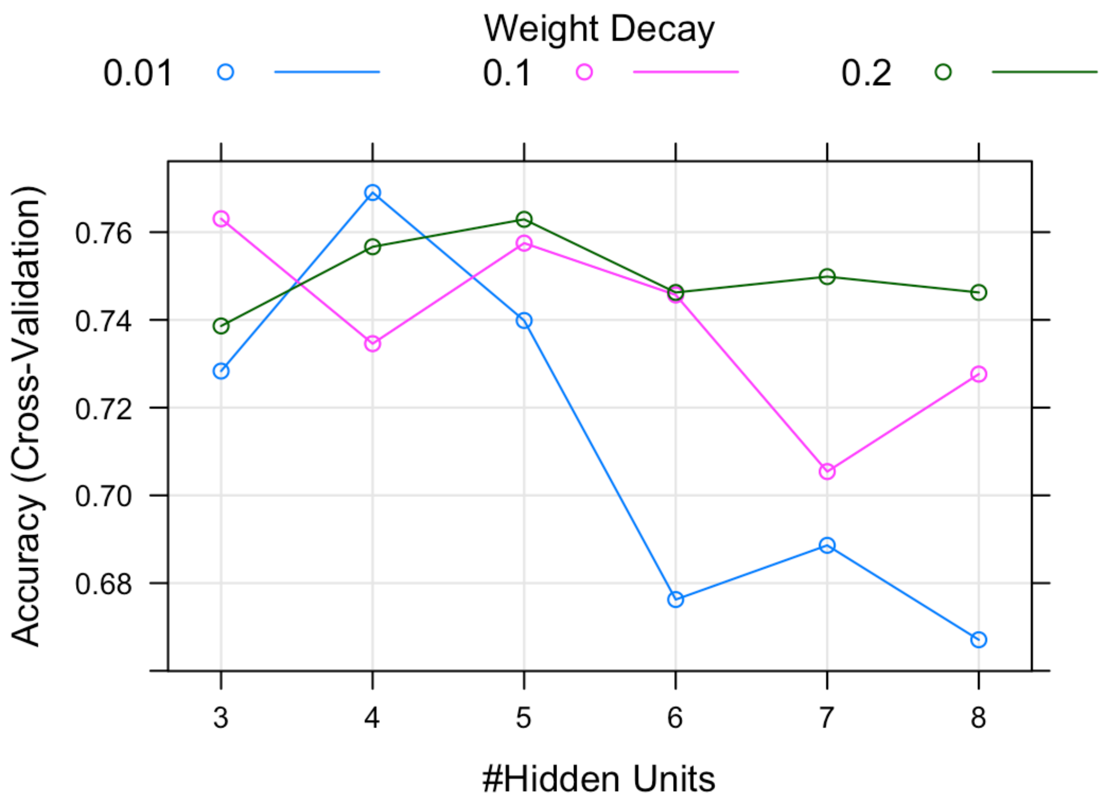
- **Learning Rate (η)**:
  - Controls the step size during the optimization process.
  - Affects how quickly the model converges to a minimum (too high can overshoot, too low can stall).
- **Regularization Parameters**:
  - **L2 Regularization (Ridge)**: Penalizes large weights, encouraging simpler models.
  - **L1 Regularization (Lasso)**: Can lead to sparse models by zeroing out weights.
- **Number of Layers**:
  - **Depth of Network**: The number of hidden layers in the network.
  - More layers can capture more complex patterns but increase computational complexity and training time.
- **Number of Neurons per Layer**:
  - Determines the capacity of each hidden layer.
  - More neurons can model more complex functions but may increase risk of overfitting.
- **Activation Functions**:
  - **ReLU (Rectified Linear Unit)**: Commonly used, helps mitigate the vanishing gradient problem.
  - **Sigmoid**: Squashes input between 0 and 1, useful for binary classification.
  - **Tanh**: Squashes input between -1 and 1, zero-centered.

## Nnet – hyperparameter tuning

```
train_control <- trainControl(
  method = "cv",
  number = 20,
  classProbs = TRUE)

tune_grid <- expand.grid(
  size = (3:8),
  decay = c(0.01,0.1,0.2))
```

# Nnet – hyperparameter tuning

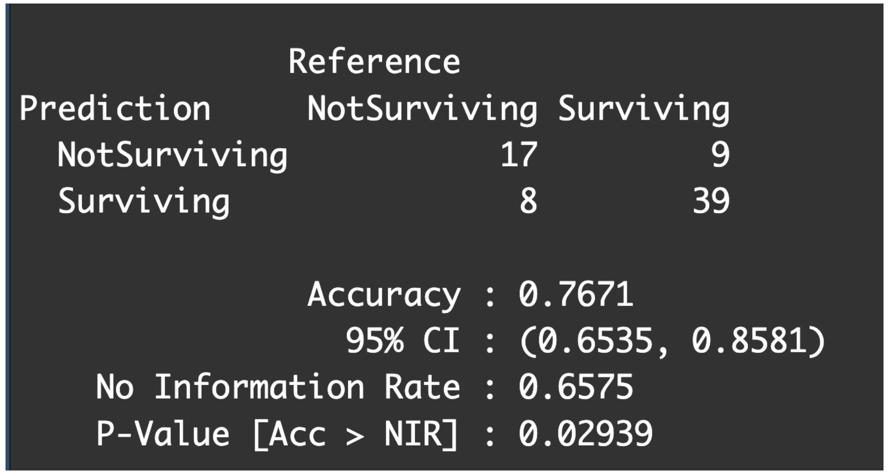
```
mod_nnet = caret::train(labels~ .,
             data=pp_train,
             method = "nnet",
             trControl = train_control,
             tuneGrid = tune_grid,
             trace = FALSE)


plot(mod_nnet)
```



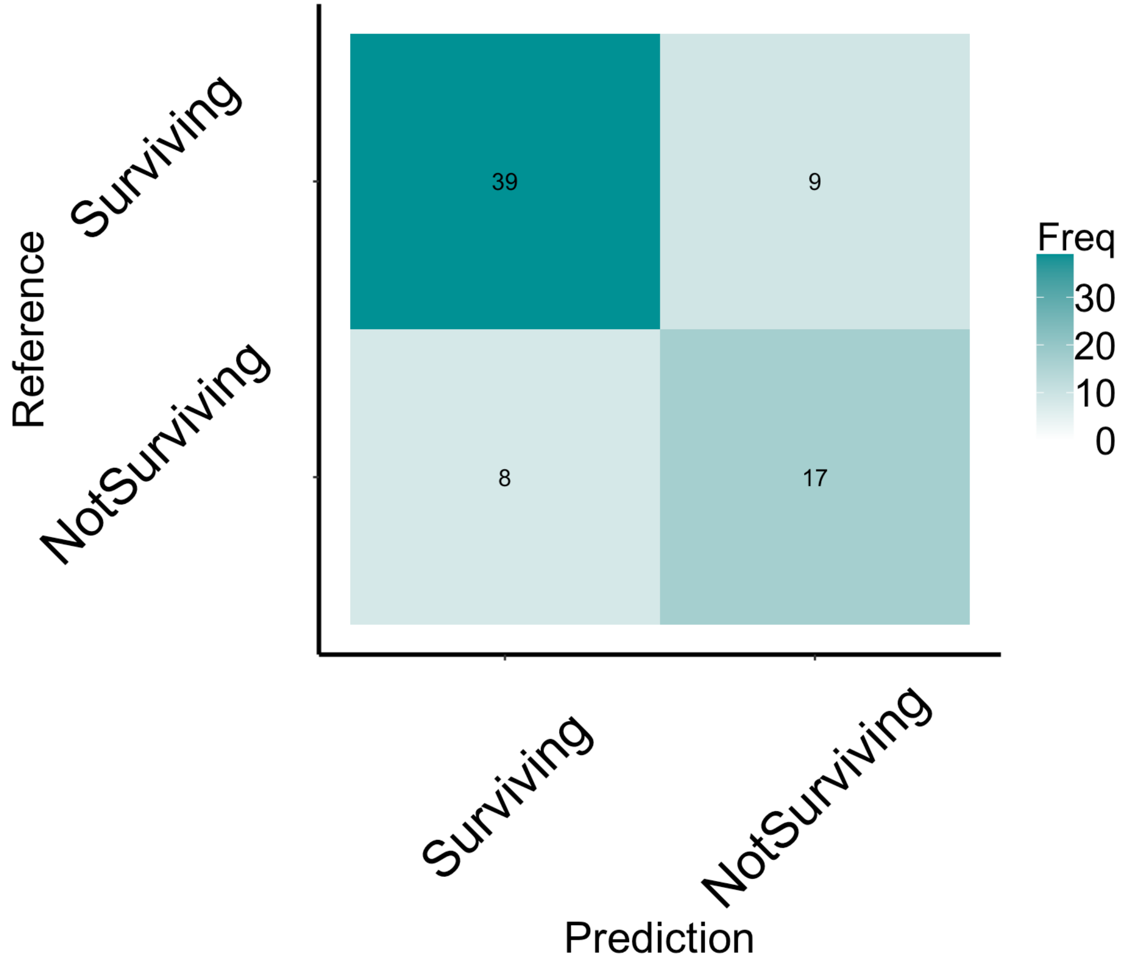
# Nnet – performance

```
pred <- predict(mod_nnet, pp_test)
```

```
(cm <- confusionMatrix(pred,
pp_test$labels))
```



```
              Reference
Prediction     NotSurviving Surviving
  NotSurviving           17         9
  Surviving               8        39

               Accuracy : 0.7671
                 95% CI : (0.6535, 0.8581)
    No Information Rate : 0.6575
    P-Value [Acc > NIR] : 0.02939
```



# Practicals 3.1

Trying different algorithms

## Balancing Data

- Data imbalance occurs when one class of the target variable is significantly more frequent than others.
- Techniques like oversampling, undersampling, SMOTE (Synthetic Minority Over-sampling Technique), and class weights can help address data imbalance.
- Balancing data enhances the model's ability to learn from minority class examples and improves overall performance.

## Loss Functions

- A function that measures the model's prediction accuracy by comparing predicted values with actual values during training.

- Common loss functions include:
  - Residual Sum of Squares (RSS),
  - Mean Squared Error (MSE),
  - Cross-Entropy Loss,
  - L1 Loss (Mean Absolute Error)
  - …

- Optimizing the loss function helps in improving the model's performance.

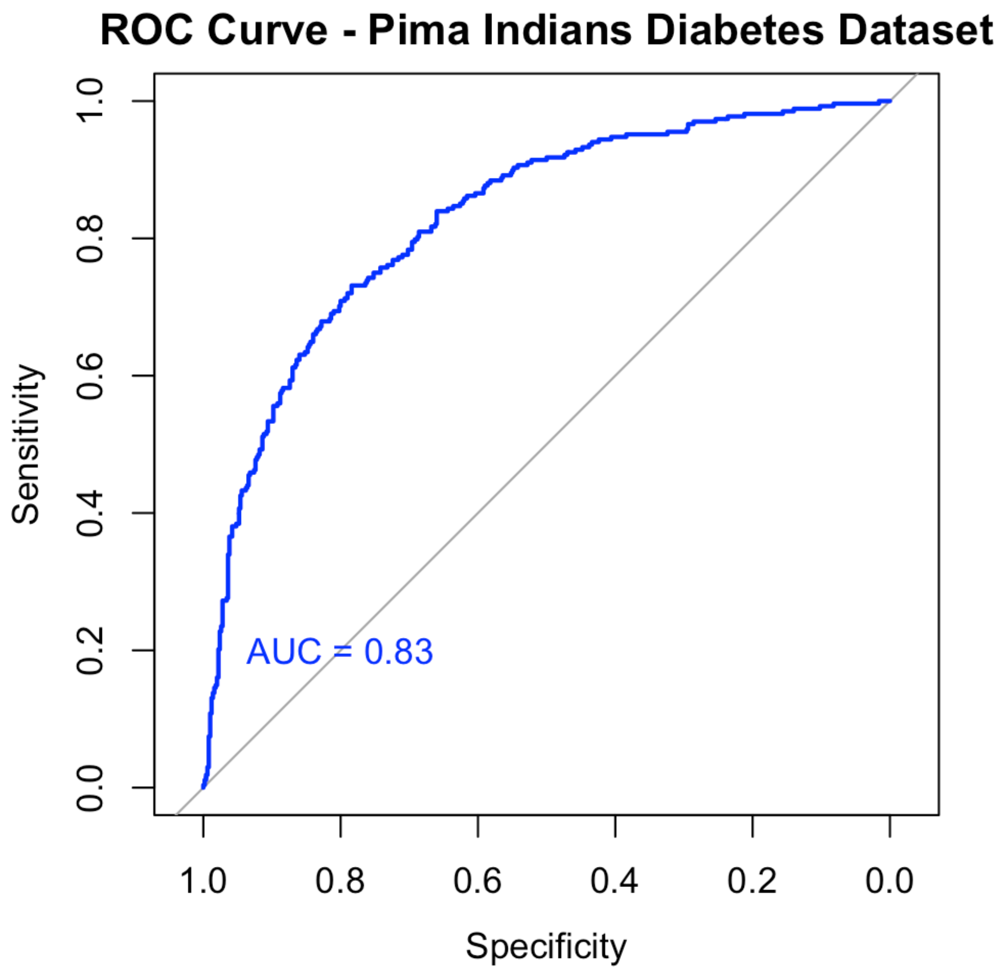
# ROC Curves

(Receiver Operating Characteristic)

- Graphical representation of the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity) for varying classification thresholds.

- Summarizes the model's performance across different threshold values.

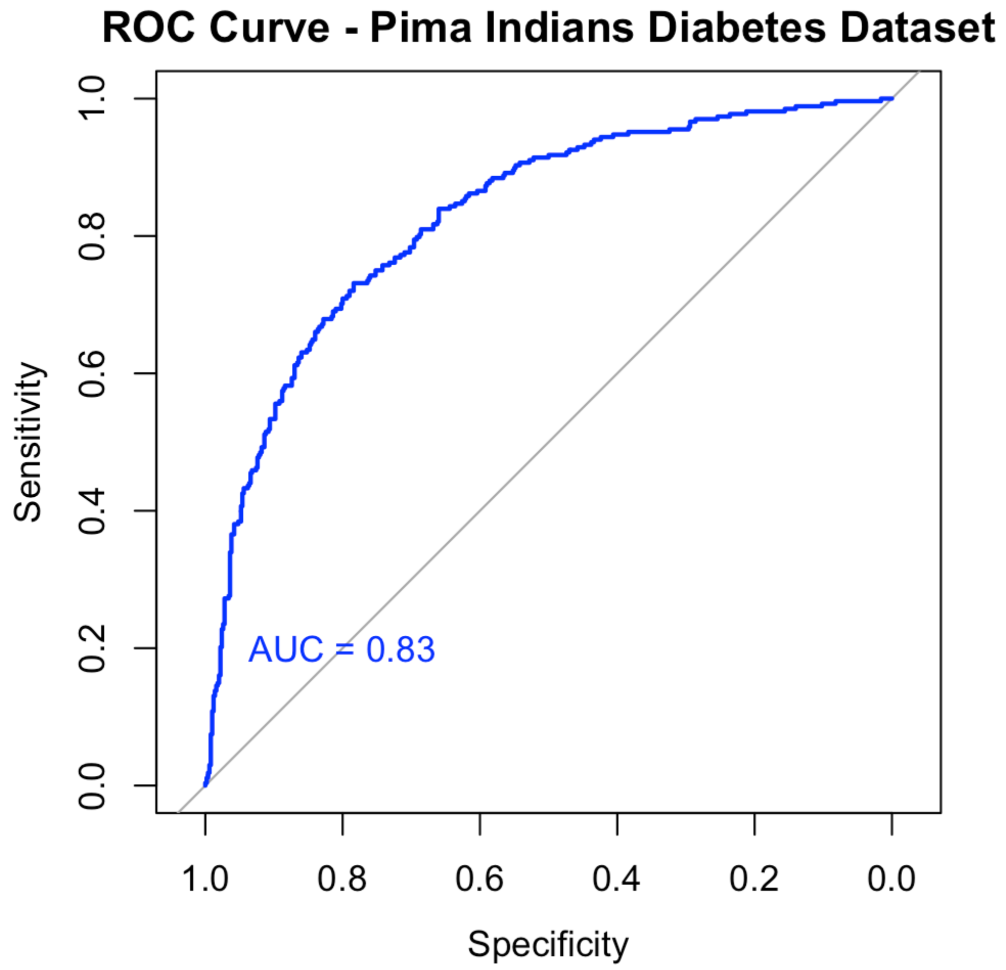
- AUC (Area Under the Curve) is often used to quantify the ROC curve's performance.



# ROC Curves

(Receiver Operating Characteristic)

```
# Make predictions on the training set
predictions <- predict(model,
                       PimaIndiansDiabetes,
                       type = "prob")


resp = ifelse(
PimaIndiansDiabetes$diabetes == "pos",
 1, 0)

# Calculate ROC curve
roc_curve <- roc(response = resp,
                 predictor = predictions$pos)
```



## Supervised Learning

- Involves training a model on labeled data where the input features and their corresponding output labels are provided.
- The model learns to map inputs to outputs by finding patterns and relationships in the labeled data.
- Requires a clear objective, such as classification or regression, to minimize a specific loss function during training.
- Examples include linear regression, logistic regression, support vector machines, and neural networks.

## Unsupervised Learning

- Involves training a model on unlabeled data where only input features are provided without explicit output labels.
- The model learns to find hidden patterns, structures, or relationships in the data without specific guidance on what to look for.
- Often used for tasks like clustering, dimensionality reduction, and anomaly detection.
- Does not rely on a predefined objective or labeled data, allowing for more exploratory data analysis and pattern discovery.

1. Clustering:
    1. K-means clustering
    2. Hierarchical clustering

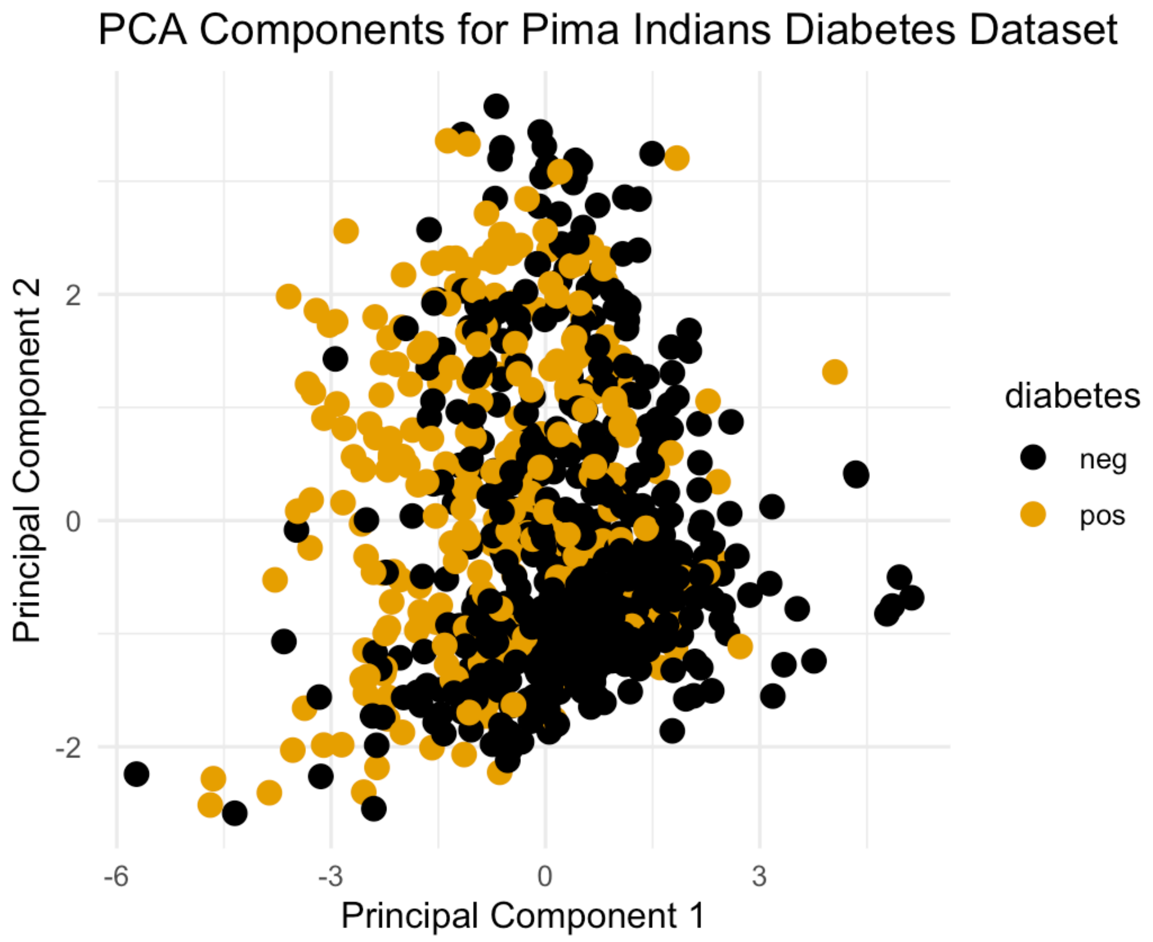
2. Dimensionality Reduction:
    1. Principal Component Analysis (PCA)
    2. t-Distributed Stochastic Neighbor Embedding (t-SNE)
    3. UMAP

3. Anomaly Detection:
    1. Isolation Forest
    2. One-Class SVM (Support Vector Machine)
    3. K-means clustering for anomaly detection

## PCA

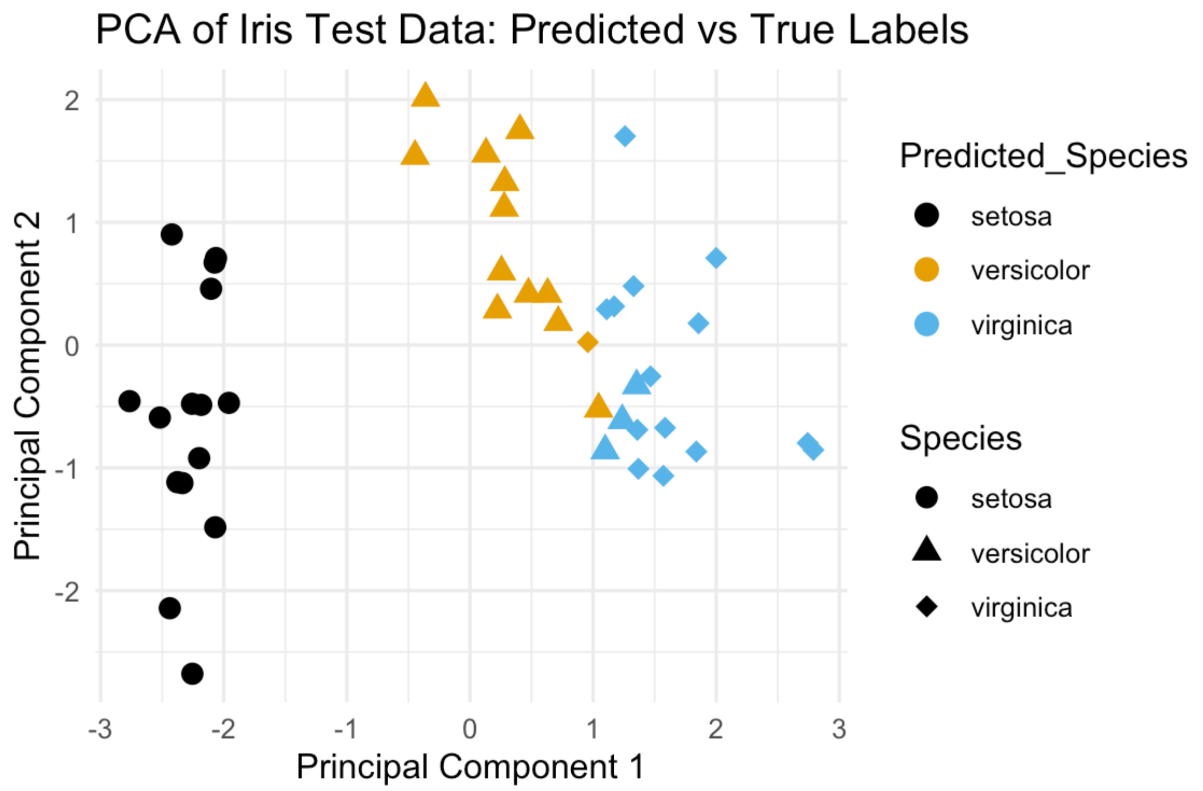
- PCA is very simple in caret:

```
preprocess_method <- preProcess(
data,method = c("pca"))


data_pca <-
predict(preprocess_method, data)
```



# PCA + Neural Network

We can do “automatic feature selection” by doing a PCA and then fit a ML model to the first *k* components:



# Practical Session 1.3

A first small project