Machine Learning in R

R-Ladies Colombo

Kasun Bandara

29 March, 2021

Melbourne Centre for Data Science, University of Melbourne, Australia.

Machine Learning in R

Kasun Bandara
University of Melbourne
Australia

Organized by

R Ladies

R Ladies Colombia

Introduction

About me

- 2015 Graduated in Computer Science from University of Colombo School of Computing
- 2015 Joined WSO2 Inc. as a Software Engineer
- 2016-2020 Ph.D. in Computer Science, Monash University, Australia
    - Topic: Forecasting In Big Data With Recurrent Neural Networks
    - Machine Learning for Time Series Forecasting
    - Research Internship at Walmart Labs, San Francisco, USA
    - Research Scientist at Turning Point, Melbourne, Australia
    - Data Science Tutor, Faculty of IT, Monash University
- 2021 Research Fellow, University of Melbourne

About me (2)

- Research Interests
  - Global Forecasting Models
  - Hierarchical Forecasting
  - Retail sales/demand forecasting
  - Renewable energy production forecasting (solar)
- Competition Fanatic !
  - M5 Forecasting Competition (**Gold Medalist**)
  - IEEE CIS Energy Forecasting Competition (**4th Place**)
  - Air-Liquide Energy Forecasting Competition (**4th Place**)
  - ANZ Customer Segmentation Challenge (**Top Performer**)

What is Data Science ?

Data Science is an interdisciplinary field that permits you to extract information from organized or unstructured data.

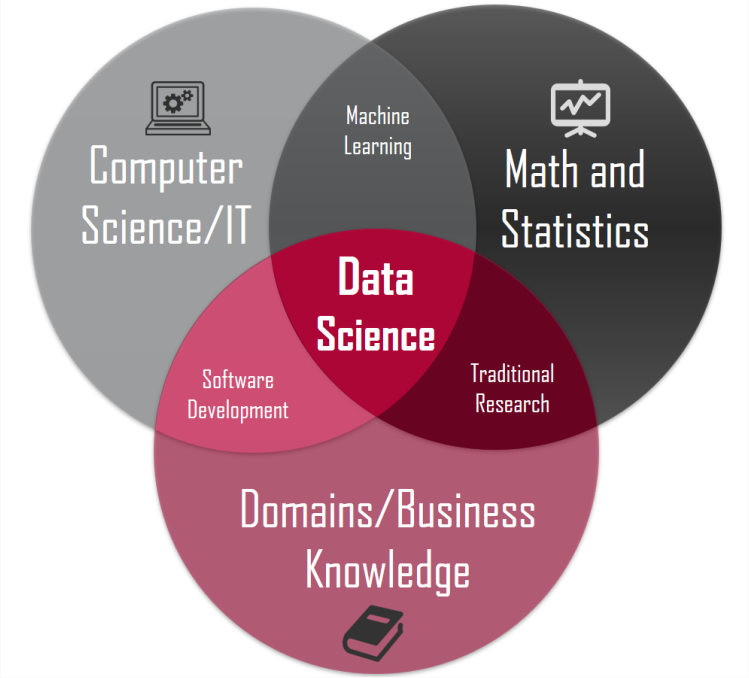


Figure 1: An intersection of many fields of science[1]

[1] Image source: https://medium.com/believing-these-8-myths-about-what-is-data-science-keeps-you-from-growing-528f1bd240dc

Data Science Life Cycle

Known as the O.S.E.M.N. framework.

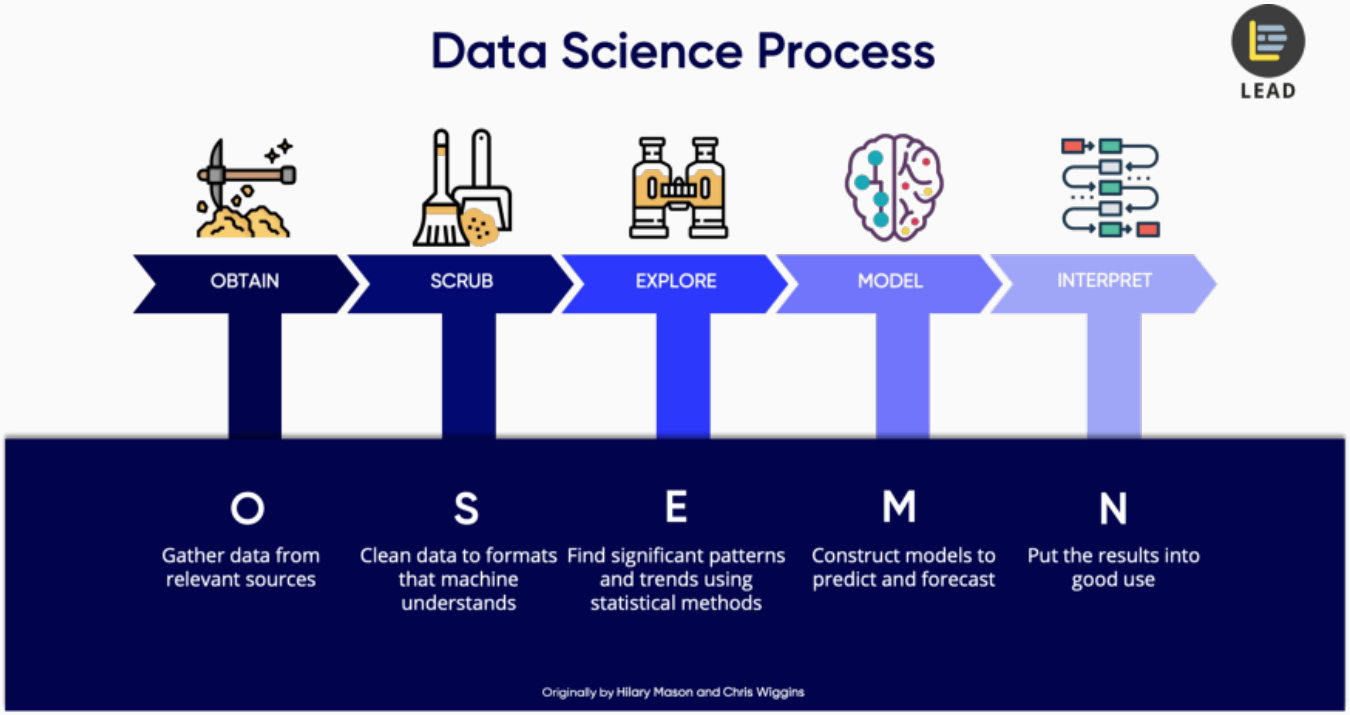


Figure 2: Data Science Process[2]

[2] Image source: https://towardsdatascience.com/5-steps-of-a-data-science-project-lifecycle-26c50372b492

Obtain (O)

- Retrieving data from multiple sources of inputs.
    - Structured Data: RDBMS, Tabular Data, CSV, TSV.
    - Unstructured Data: NoSQL Databases, API Data (Twitter, Facebook).
- Databases: `{odbc}`
- Scraping data from websites: `{rvest}`
- Data platforms: **Kaggle**, **UCI**, **Competition Datasets**, **Government APIs**

Example of {rvest}

```
library(rvest)
library(dplyr)
set.seed(1234)

# reading the HTML page (Lord of the Rings)
lor_movie <- read_html("https://www.imdb.com/title/tt0120737/")

# Scraping the movie rating.
lor_movie %>%
  html_node("strong span") %>%
  html_text() %>%
  as.numeric()
#[1] 8.8

# Scraping the cast.
lor_movie %>%
  html_nodes("#titleCast .itemprop span") %>%
  html_text()

# Scraping the movie poster.
lor_movie %>%
  html_nodes("#img_primary img") %>%
  html_attr("src")
```

Scrub (S)

- Also known as **data pre-processing**, **data wrangling**.
- Converting the data into a unified, suitable format
  - Easier for the data exploration process.
  - What your predictive algorithm expects ?
  - **tidyverse**

    `{dplyr,tidyr,stringr,tibble,purr,ggplot2}`
- Handles data issues
  - Cleaning: Missing values, Outliers, Noisy data.
  - Transformation: Normalisation, Feature Discretization.
  - Reduction: Feature selection, Dimensionality reduction.

Missing Value Imputation

```r
library(simputation)
set.seed(1234)

# Loading iris dataset and randomly inserting NAs.
df <- iris
df_NA <- as.data.frame(lapply(df, function(imp) imp[ sample(c(TRUE, NA),
        prob = c(0.85, 0.15), size = length(imp), replace = TRUE)]))

# Using median to impute the missing values.
median_imputed <- impute_median(df_NA,
                                Sepal.Length ~ Species)

# Using linear regression to impute the missing values.
linear_imputed <- impute_lm(df_NA, Sepal.Length ~ Sepal.Width + Species)

# Using CART algorithm to impute the missing values.
cart_imputed <- impute_cart(df_NA, Species ~ .)

# Imputing multiple variables at once.
multivariable_imputed <- impute_rlm(df_NA, Sepal.Length + Sepal.Width
                                    ~ Petal.Length + Species)

# Imputing using a pre-trained model.
model <- lm(Sepal.Length ~ Sepal.Width + Species, data=iris)
model_imputed <- impute(df_NA, Sepal.Length ~ model)
```

Dealing with Outliers

- A data point that differs significantly from other observations.
- Observations that distort your analysis.
  - Boxplot visualisation: `{ggplot2}`
  - Grubbs's test, Dixon's test, Rosner's test: `{outliers}`
  - Outlier detection algorithms: `{OutlierDetection}`
  - **outlierTest()** from `{car}`
  - **lofactor()** from `{DMwR}` (Local Outlier Factor)
- Anomaly detection is itself a different research area !
  - One Class SVM, IsolationForest
  - Unsupervised algorithms (Clustering)
  - Time series: `{tsoutliers,oddstream,stray}`

Feature Selection

- Removing redundant features from the dataset.
- Computational complexity, Address model overfitting.
- **Filter Methods**
  - Features are selected based on a statistical score.
  - Independent of any machine learning algorithm.
  - **Pearson's Correlation, Chi-Square, PCA**
- **Wrapper Methods**
  - A subset of features are used to train a model.
  - Forward, Backward, Recursive elimination.
  - Inbuilt penalization functions: **LASSO, RIDGE** regression
  - `{Boruta,caret,glmnet}`

Using Correlation

```
library(GGally)
```

```
## Loading required package: ggplot2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

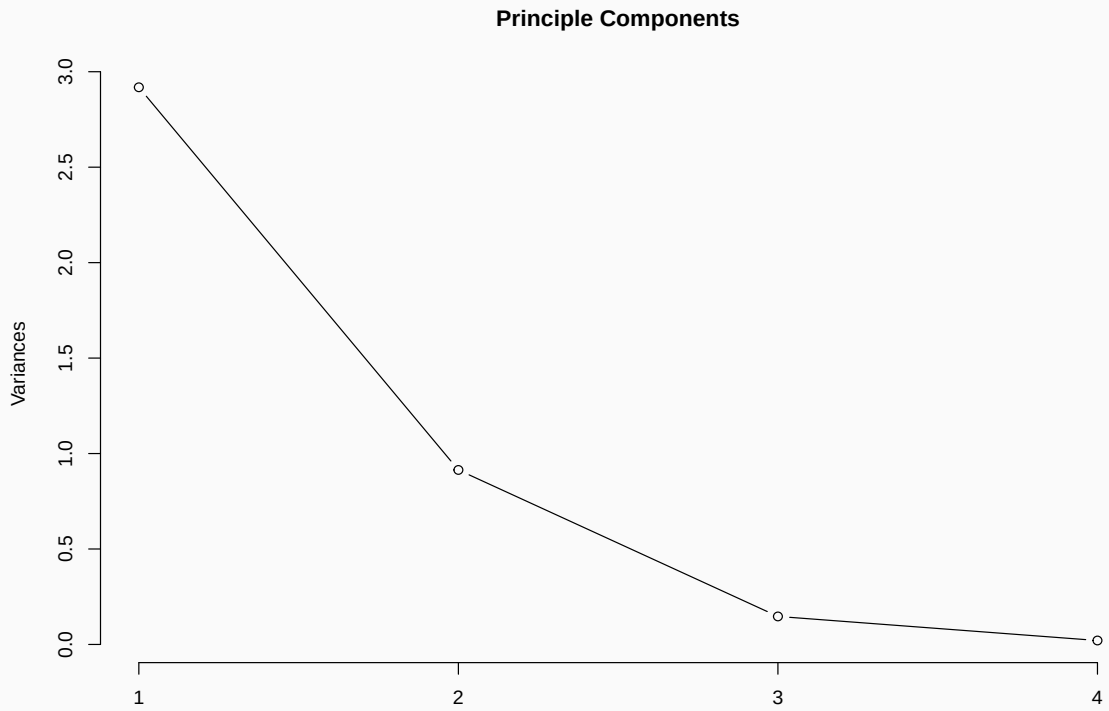
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
set.seed(1234)

# Plotting the feature correlations.
iris %>% select(-Species) %>% ggpairs()
```

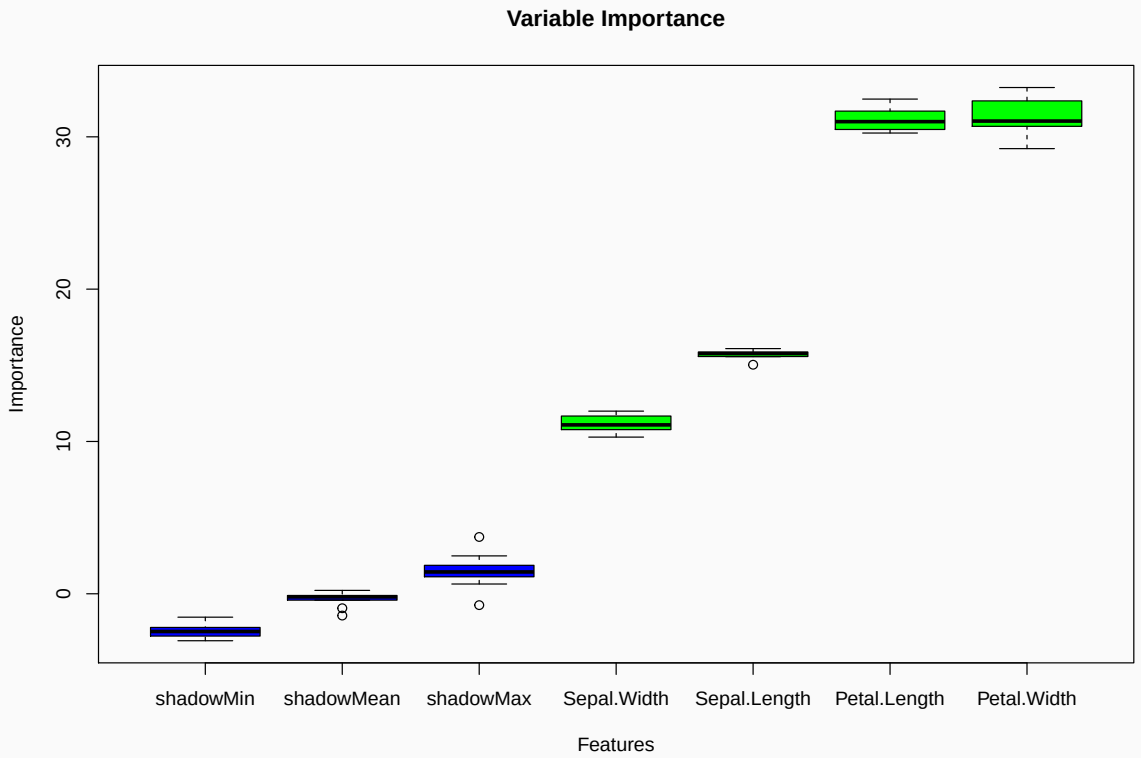
Using PCR

```r
library(dplyr)
set.seed(1234)

# Plotting the feature importance.
pcomp_df <- iris %>%
  select(-Species) %>% prcomp(scale. = T, center = T) %>%
  plot(type="l", main = "Principle Components")
```



Example of {Boruta}

```r
library(Boruta)
set.seed(1234)

# Boruta is a feature selection algorithm based on the random forests algorithm.
boruta_df <- Boruta(Species ~ ., data=iris, doTrace=0)

# Plotting the feature importance.
plot(boruta_df, xlab="Features", main="Variable Importance")
```

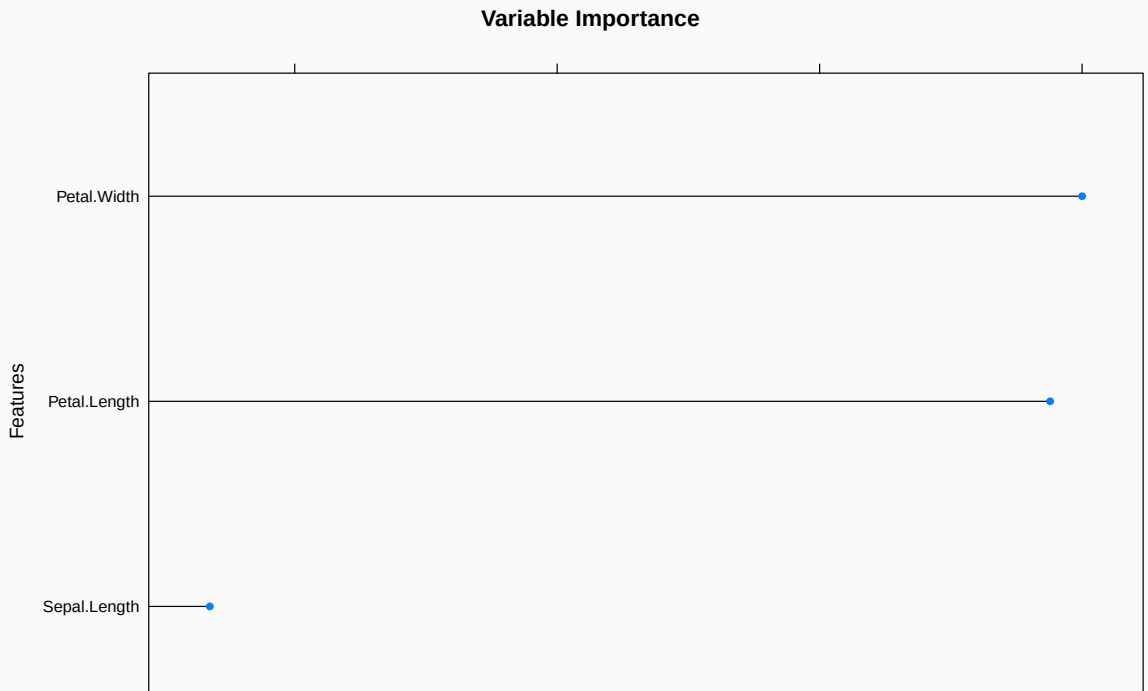


Example of {caret}

```
library(caret)
```

```
## Loading required package: lattice
```

```
set.seed(1234)

# Build a decision tree model using rpart (Recursive Partitioning And Regression Trees)
rPart_df <- train(Species ~ ., data=iris, method="rpart")
rPart_imp <- varImp(rPart_df)

# Plotting the feature importance.
plot(rPart_imp, top = 3, main='Variable Importance', ylab = "Features")
```



Explore (E)

- Examination of data, features, and their characteristics.
    - Data types: numerical, ordinal, and nominal data.
    - Summary statistics.
    - Feature distributions.
    - Feature correlations (positive, negative).
    - Classification: class distribution (**Class Imbalance?**)
- Invest your time more on the data exploration process.
    - Frequency distribution: **Histograms**
    - Outlier detection: **Box plots**
    - Feature correlation analysis: **Scatter plots**
    - Time series analysis: **Trend and Seasonal plots**

Tools available for Exploration

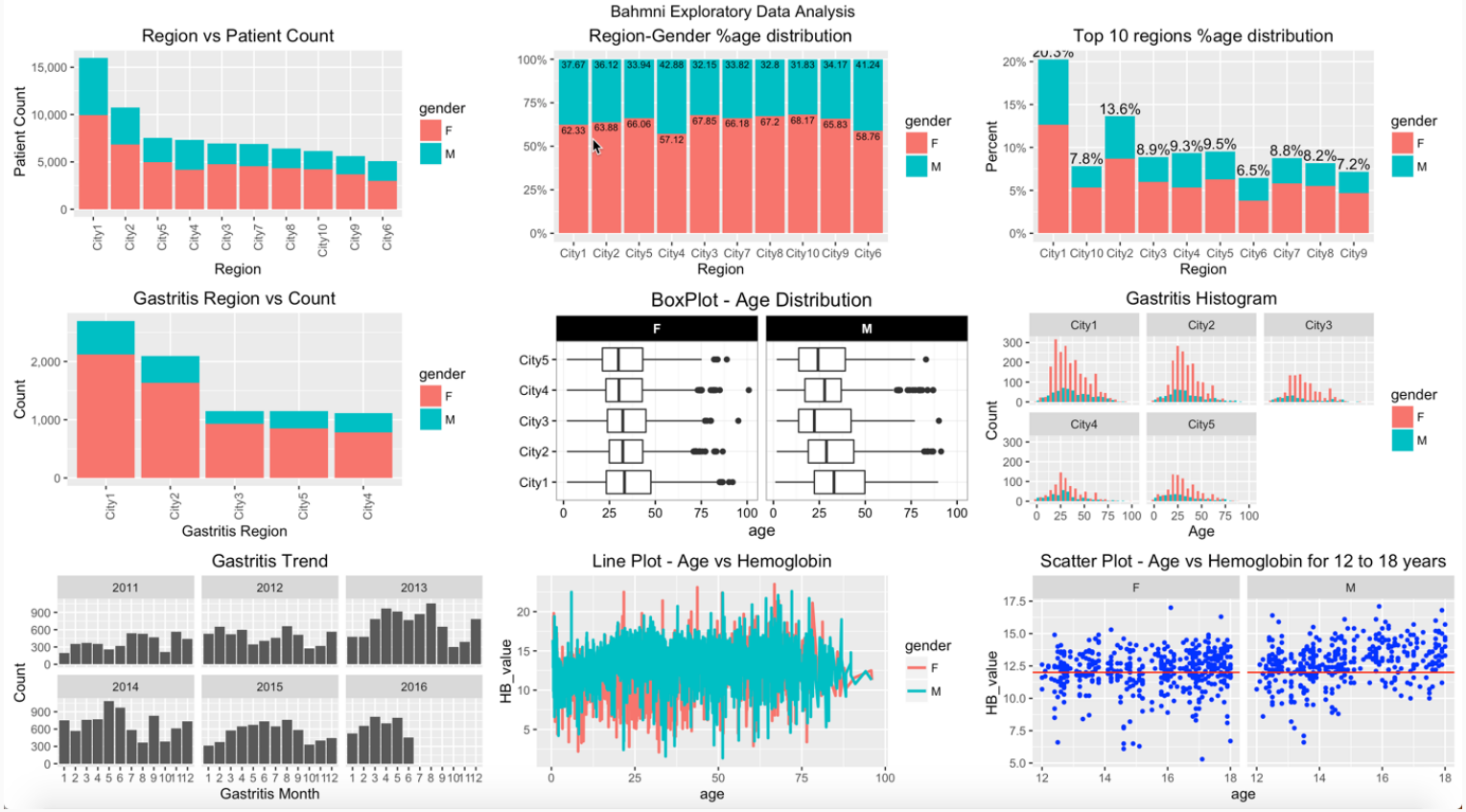


Figure 3: Plots available from `{ggplot2}`[3]

[3] Image source: https://www.pinterest.com.au/pin/281686151677624808/

Seasonal plot from {feasts}

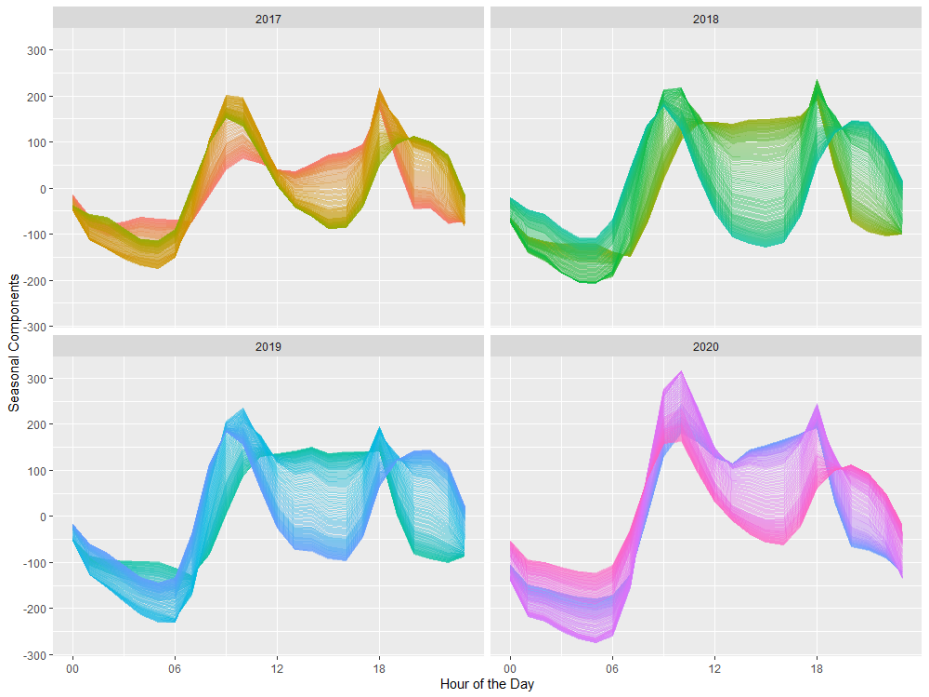


Figure 4: The presence of multiple seasonal cycles[4]

[4] Github repo: https://github.com/kasungayan/Meldatathon2020

Model Development (MD)

- Model parameter estimation, Hyper-parameter tuning.

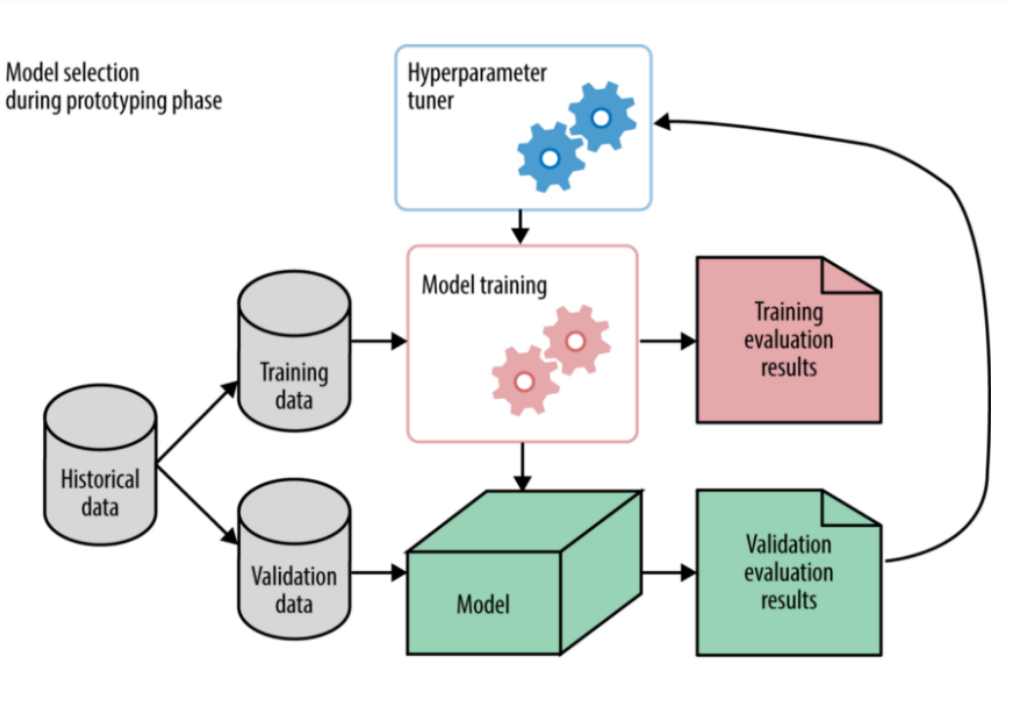


Figure 5: Model Training and Validation[5]

[5] Image source: https://towardsdatascience.com/5-steps-of-a-data-science-project-lifecycle-26c50372b492

Model Development Techniques

- **Supervised Learning** or **Unsupervised Learning**
  - Regression: Linear-regression, Support Vector Machine (SVM), Lasso-Regression
  - Classification: Naive Bayes, Random Forest, K-Nearest Neighbors
  - Clustering: K-Means, Fuzzy C-Means, Self Organising Maps (SOM)
- Neural Networks: Multilayer Perceptron (**MLP**), Recurrent Neural Network (**RNN**), Convolutional Neural Network (**CNN**)
- Different applications: **Spam Detection**, **Market Segmentation**, **Image Classification**, **Time Series Forecasting**, **Language Translation**, **Voice Recognition**

Naive Bayes classifier

```r
library(mlbench) # multiple benchmark datasets for different machine learning tasks.
library(caret) # multiple inbuilt regression and classification algorithms.
library(rsample) # data splitting.
library(dplyr)

data(BreastCancer)
set.seed(1234)

# Splitting the data into train and test sets.
df_BreastCancer_split <- initial_split(BreastCancer, prop = .7)
# Similar splitting using caTools
# df_BreastCancer_split <- sample.split(BreastCancer,SplitRatio = 0.7)

df_BreastCancer_train <- training(df_BreastCancer_split)
df_BreastCancer_test  <- testing(df_BreastCancer_split)

# Checking for class distribution.
table(df_BreastCancer_train$Class) %>% prop.table()
```

```
##
##    benign malignant
## 0.6571429 0.3428571
```

```r
table(df_BreastCancer_test$Class) %>% prop.table()
```

```
##
##    benign malignant
## 0.6507177 0.3492823
```

Naive Bayes classifier Cont.

```r
# create feature and class attributes.
features <- setdiff(names(df_BreastCancer_train), "Class")
train_features <- df_BreastCancer_train[, features]
train_class <- df_BreastCancer_train$Class

# Define a 10-fold cross validation procedure.
train_control <- trainControl(method = "cv", number = 10)

# train the naive bayes model
model_nb1 <- train(x = train_features, y = train_class, method = "nb", trControl = train_control)
#Show the confusion matrix.
confusionMatrix(model_nb1)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  benign malignant
##   benign      63.5       0.4
##   malignant    2.2      33.9
##
##  Accuracy (average) : 0.9735
```

Naive Bayes classifier Cont.

```r
# hyper-parameter grid
hyper_search_grid <- expand.grid(usekernel = c(TRUE, FALSE), fL = 0:5, adjust = seq(0, 5, by = 1))

# train the naive bayes model using a hyper-parameter grid.
model_nb2 <- train(x = train_features, y = train_class, method = "nb",
                   trControl = train_control, tuneGrid = hyper_search_grid)

# Printing best models.
# model_nb2$results %>%
#  arrange(desc(Accuracy)) %>% head(4)

# results for best model
confusionMatrix(model_nb2)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  benign malignant
##   benign      63.5       0.8
##   malignant    2.2      33.5
##
##  Accuracy (average) : 0.9694
```

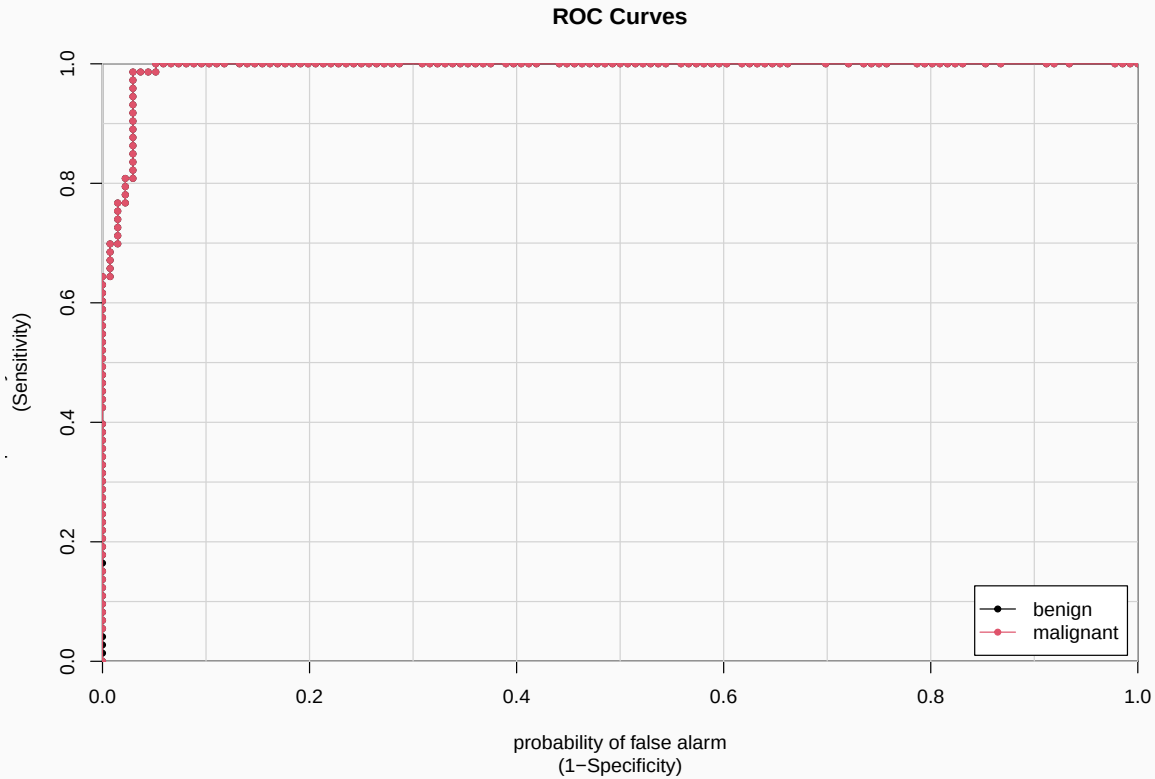
Naive Bayes results on the testset

```r
# Applying the best model to unseen (test) dataset.
prediction_test <- predict(model_nb2, newdata = df_BreastCancer_test)
# Printing the confusing matrix on the test set.
confusionMatrix(prediction_test, df_BreastCancer_test$Class)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign       132         1
##   malignant      4        72
##
##                Accuracy : 0.9761
##                  95% CI : (0.9451, 0.9922)
##     No Information Rate : 0.6507
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9479
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.9706
##             Specificity : 0.9863
##          Pos Pred Value : 0.9925
##          Neg Pred Value : 0.9474
##              Prevalence : 0.6507
##          Detection Rate : 0.6316
##    Detection Prevalence : 0.6364
##       Balanced Accuracy : 0.9784
```

Generating the ROC curve

```r
library(caTools) #to generate ROC curves

prob_results <- predict(model_nb2, df_BreastCancer_test, type = "prob")

# Generating the ROC curve for the test set.
caTools::colAUC(prob_results, df_BreastCancer_test[["Class"]], plotROC = TRUE)
```




```
##                          benign malignant
## benign vs. malignant 0.9917405 0.9917405
```

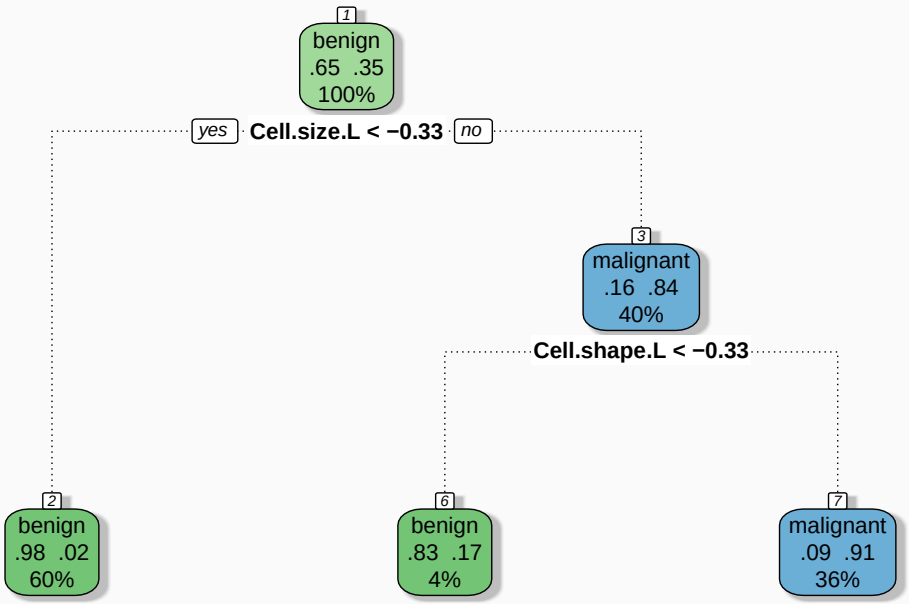
Decision tree

```r
library(caret)
library(rpart)

set.seed(1234)

# Remove incomplete records.
df_BreastCancer_train <- df_BreastCancer_train[complete.cases(df_BreastCancer_train), ]

# Define a 10-fold cross validation procedure.
train_control <- trainControl(method = "cv", number = 10)

# train the naive bayes model.
model_dt <- train(Class ~ ., data=df_BreastCancer_train, method = "rpart", trControl = train_control)
#Show the confusion matrix.
confusionMatrix(model_dt)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  benign malignant
##   benign      62.2       3.3
##   malignant    3.1      31.3
##
##  Accuracy (average) : 0.9357
```

Visualising the decision tree

```
library(rattle)
set.seed(1234)
# Generating the decision tree.
# You can also separately use the rpart and rpart.plot to reproduce this.

fancyRpartPlot(model_dt$finalModel)
```



Rattle 2021−Mar−23 01:36:41 kasun

Regression using {caret}

```r
data(diamonds)
set.seed(1234)

# Data splitting.
data_split <- initial_split(diamonds, prop = .7)
train <- training(data_split)
test  <- testing(data_split)

# Linear regression model training.
model <- train(price ~ ., train, method = "lm",trControl = trainControl(method = "cv",
                                        number = 10))
print(model)
```

```
## Linear Regression
##
## 37758 samples
##     9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 33982, 33982, 33983, 33982, 33982, 33981, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1126.445  0.9204881  742.5132
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Regression using {glmnet}

```
library(glmnet)
library(dplyr)

data(diamonds)
set.seed(1234)
# Defining the feature variables.
x <- model.matrix(price~., train)[,-1]
# Defining the class variable.
y <- train$price

# Use cross validation to determine the optimal lambda.
cv <- cv.glmnet(x, y, alpha = 1)
# Fit the final model (lasso regression) on the training data
best_model <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)

# Evaluating the model on the test data.
x.test <- model.matrix(price~ ., test)[,-1]
price_predictions <- best_model %>% predict(x.test) %>% as.numeric()
price_predictions[price_predictions < 0] <- 0

# Model performance summary.
data.frame(
  RMSE = RMSE(price_predictions, test$price),
  Rsquare = R2(price_predictions, test$price)
)
```

```
##       RMSE   Rsquare
## 1 1088.945 0.9254219
```

Regression using {glmnet}

```
library(glmnet)
library(dplyr)

data(diamonds)
set.seed(1234)
# Defining the feature variables.
x <- model.matrix(price~., train)[,-1]
# Defining the class variable.
y <- train$price

# Use cross validation to determine the optimal lambda.
cv <- cv.glmnet(x, y, alpha = 1)
# Fit the final model (lasso regression) on the training data
best_model <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)

# Evaluating the model on the test data.
x.test <- model.matrix(price~ ., test)[,-1]
price_predictions <- best_model %>% predict(x.test) %>% as.numeric()
price_predictions[price_predictions < 0] <- 0

# Model performance summary.
data.frame(
  RMSE = RMSE(price_predictions, test$price),
  Rsquare = R2(price_predictions, test$price)
)
```

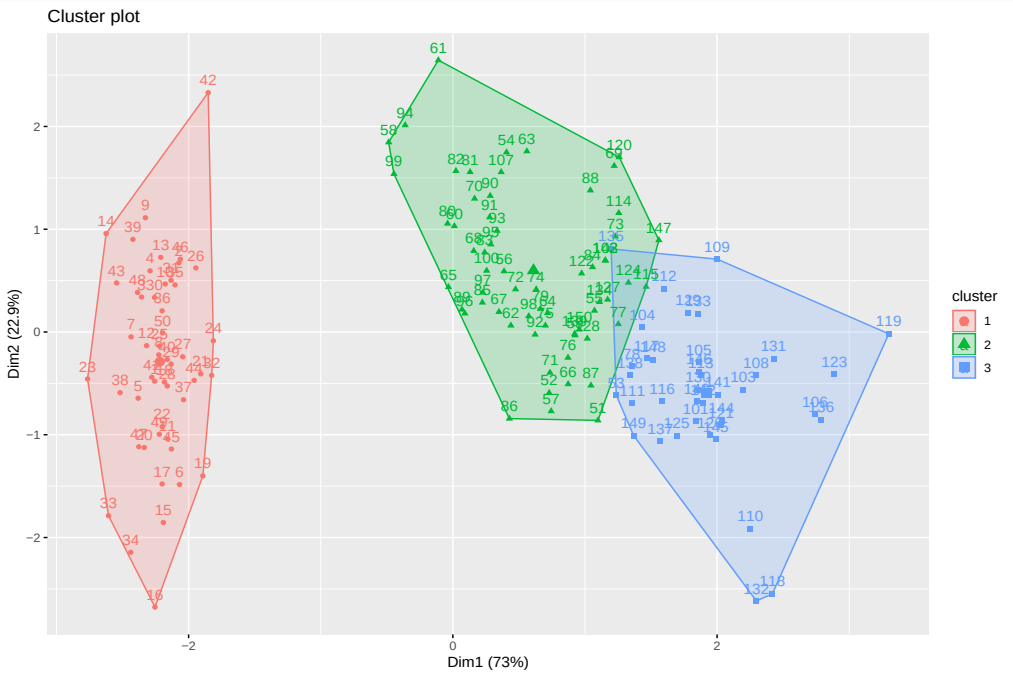
```
##       RMSE   Rsquare
## 1 1088.945 0.9254219
```

Unsupervised Learning

- Learning patterns from unlabbled data.
- Clustering ?
    - K means: Computationally efficient, **Optimal K ?, Outliers?**
    - Elbow and Silhouette methods to determine the optimal K
    - **DBSCAN**: No restrictions on the cluster shapes
    - Feature are categorical ? **Partitioning Around Medoids (PAM)**
    - Hierarchical clustering: **cluster dendrogram**
- Auto-Encoders ?

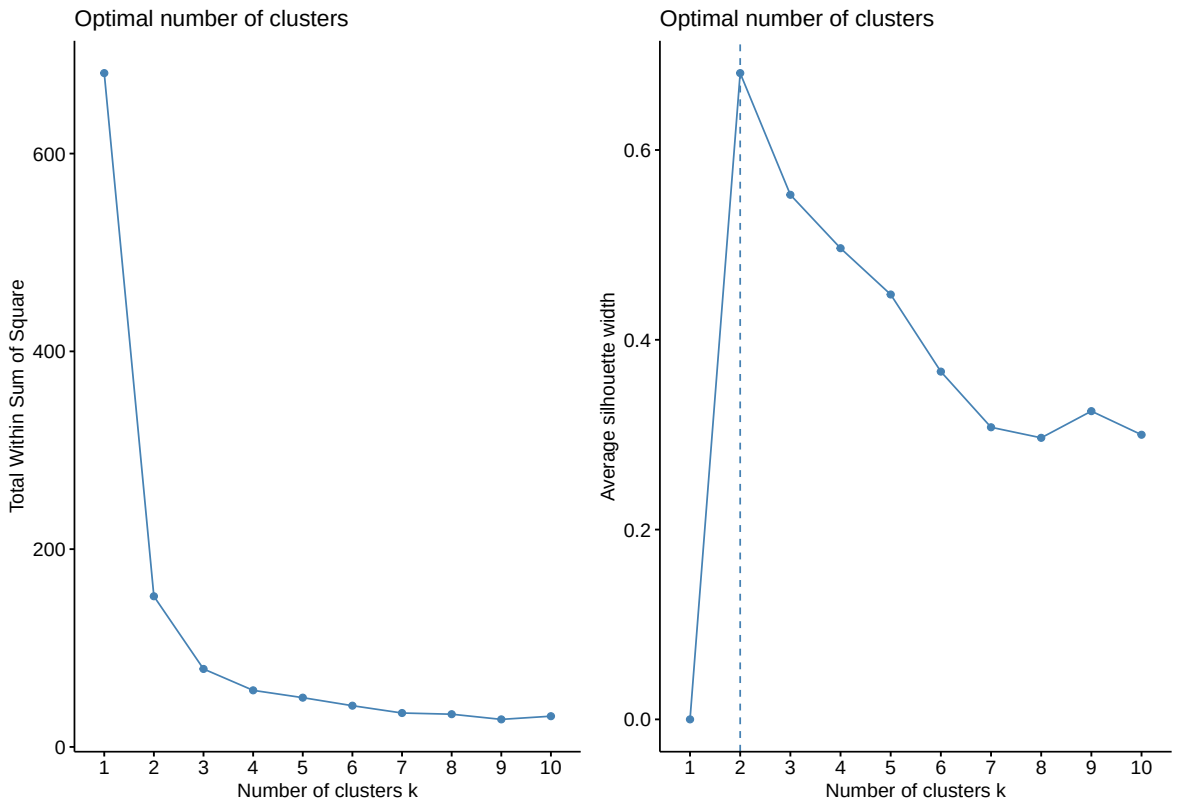
Kmeans clustering

```
library(factoextra)

set.seed(1234)

# Removing the categorical class.
df <- iris[, -5]
# Applying kmeans algorithm with k = 3.
cluster_output <- kmeans(df, centers = 3, nstart = 25)
# Illustrating the cluster distribution.
fviz_cluster(cluster_output, data = df)
```



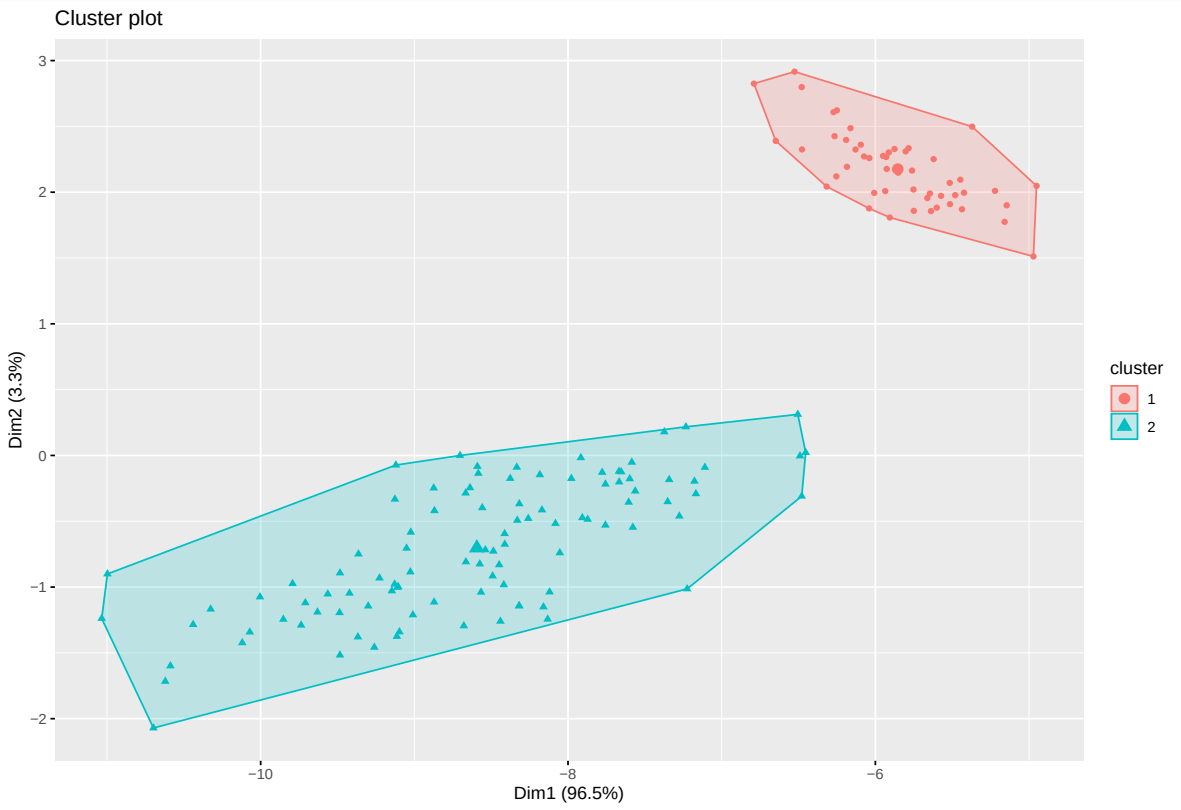
Optimal K

```
library(gridExtra)

set.seed(1234)
# Different methods to determine the optimal K.
elbow <- fviz_nbclust(df, kmeans, method = "wss")
silhouette <- fviz_nbclust(df, kmeans, method = "silhouette")

grid.arrange(elbow, silhouette, nrow = 1)
```



Clustering using DBSCAN

```r
library(factoextra)
library(fpc)

set.seed(1234)
df <- iris[, -5]

# Using DBSCAN algorithm without setting the K.
db_output <- fpc::dbscan(df, eps = 0.95, MinPts = 5)
fviz_cluster(db_output, df, stand = FALSE, frame = FALSE, geom = "point")
```



Neural Networks

- Strong computational systems that mimic human brain.
- Hold the universal approximation property.
- Backpropagation algorithm for training.
- State-of-the-art for many prediction/classification applications.
- Different variants of neural networks.
    - Multi-Layer Perceptrons (MLP): `{nnet,neuralnet}`
    - Recurrent Neural Networks (RNN): `{rnn,RSNNS}`
    - Convolutional neural networks (CNNs)
- **Tensorflow, Keras, Torch** APIs in R

Feedforward Neural Networks using {nnet}

```
library(caret)

data(diamonds)
set.seed(1234)

# Preprocessing.
# Use model.matrix for categorical variables (one-hot encoding)
df_diamonds <- diamonds[, c(-2, -3, -4)]

# Data splitting.
data_split <- initial_split(df_diamonds, prop = .7)
train <- training(data_split)
test  <- testing(data_split)

# You can directly use the nnet function from the nnet packaage
# mlp_fit <- nnet(price ~ ., train, size = 2, rang = 0.1, decay = 5e-4, maxit = 200)

# You can use the nnet as the method in the caret function.
mlp_fit <- train(price ~ ., train, method = "nnet",trControl = trainControl(method = "cv",
                                        number = 10))
#print(mlp_fit)
```

Deep Neural Netowrks using {keras}

```
library(keras)
library(rsample)
library(ggplot2)
library(dplyr)

data(diamonds)
set.seed(1234)

# The initial preprocessing.
# Use model.matrix for categorical variables (one-hot encoding)
df_diamonds <- diamonds[, c(-2, -3, -4)]

# Data splitting.
data_split <- initial_split(df_diamonds, prop = .7)
train <- training(data_split)
test  <- testing(data_split)

features <- setdiff(names(train), "price")
train_features <- train[, features]
train_class <- train$price
```

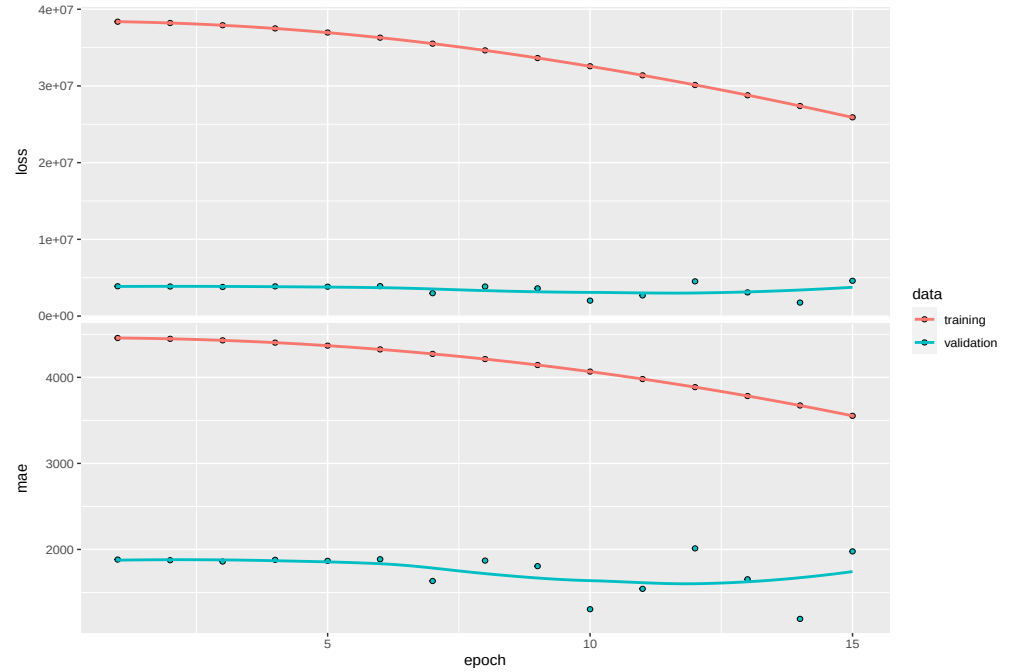
Model definition

```
# Defining the model
MLP_model <- keras_model_sequential() %>%
  # The overall neural network architecture
  layer_dense(units = 10, activation = "relu", input_shape = ncol(train_features)) %>%
  layer_batch_normalization() %>% layer_dense(units = 5, activation = "relu") %>%
  layer_batch_normalization() %>% layer_dropout(rate = 0.2) %>% layer_dense(units = 1) %>%
  # Defining the optimization algorithm and loss function.
  # Use binary and multi-categorical cross entropy for classification.
  compile(optimizer = optimizer_rmsprop(),loss = "mse",metrics = c("mae"))

# print the summary of the model
summary(MLP_model)
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense_2 (Dense)              (None, 10)                70
## _________________________________________________________________________
## batch_normalization_1 (BatchNormali (None, 10)                40
## _________________________________________________________________________
## dense_1 (Dense)              (None, 5)                 55
## _________________________________________________________________________
## batch_normalization (BatchNormaliza (None, 5)                 20
## _________________________________________________________________________
## dropout (Dropout)            (None, 5)                 0
## _________________________________________________________________________
## dense (Dense)                (None, 1)                 6
## =========================================================================
```

Model Training

```r
# train the model
learn <- MLP_model %>% fit(
  x = as.matrix(train_features),
  y = as.matrix(train_class),
  epochs = 15,
  batch_size = 32,
  validation_split = .2,
  verbose = TRUE
)

plot(learn)
```



Model Results

```
# Defining the real test features and actual test output.
test_features <- test[, features]
test_class <- test$price

# Generating predictions for the first 5 records in the test set.
model_predictions <- MLP_model %>% predict(as.matrix(test_features[1:5,]))
model_predictions[model_predictions < 0] <- 0

# Generating error summary statistic.
model_results <- MLP_model %>% evaluate(as.matrix(test_features), as.matrix(test_class))
```

Hyper-parameter optimization

- Determining the optimal hyper-parameters for a machine learning algorithm.
- Important for models with large number of hyper-parameters (neural networks)
  - Random search
  - Grid search
  - Bayesian optimization: `{rBayesianOptimization,mlrMBO}`
  - Genetic algorithm: `{GA}`
- An intelligent hyper-parameter search ?

Bayesian optimization using {rBayesianOptimization}

```
library(caret)
library(rBayesianOptimization)
set.seed(1234)

df_households <- read.csv("data/realestate.csv")
df_households_filtered <- df_households[, c(-1,-2)]
colnames(df_households_filtered)[6] <- "price"
# Data splitting.
data_split <- initial_split(df_households_filtered, prop = .7)
train <- training(data_split)
test  <- testing(data_split)
# Define a 10-fold cross validation procedure.
train_control <- trainControl(method = "cv", number = 5)
# Defining the fit of the SVM model
svm_model_fit <- function(logC, logSigma) {
  model <- train(price ~ ., data = train, method = "svmRadial", metric = "RMSE",
                 trControl = train_control,tuneGrid = data.frame(C = exp(logC), sigma = exp(logSigma)))
  list(Score = -getTrainPerf(mod)[, "TrainRMSE"], Pred = 0)
}
## Define the bounds and search for hyper-parameters.
lower_bounds <- c(logC = -8, logSigma = -10)
upper_bounds <- c(logC = 15, logSigma = -0.65)
svm_bounds <- list(logC = c(lower_bounds[1], upper_bounds[1]), logSigma = c(lower_bounds[2], upper_bounds[2]))
# svm_ba_search <- BayesianOptimization(svm_model_fit,
#    bounds = svm_bounds,init_grid_dt = NULL, init_points = 0,n_iter = 4, verbose = TRUE)
```

Light Gradient Boosting Machine (LightGBM)

- Gradient boosting framework that uses tree based learning algorithms.
- Used for both classification and regression tasks.
- Leading algorithm in many machine learning competitions (Kaggle).
    - Faster training speed and higher efficiency
    - Lower memory usage
    - Highly scalable
    - Highly parallelizable
    - Better accuracy than any other boosting algorithms

LightGBM using {lightgbm}

```
library(lightgbm)
set.seed(1234)

data("iris")
df <- iris

# Data preparation.
df$Species <- as.numeric(as.factor(df$Species)) - 1
# Data splitting.
data_split <- initial_split(df, prop = .7)
train <- training(data_split)
test  <- testing(data_split)

# Transforming to lightgbm data input format.
dtrain <- lgb.Dataset(data = as.matrix(train[1:4]), label = as.matrix(train[,5]))
dtest <- lgb.Dataset.create.valid(dtrain, data = as.matrix(test[, 1:4]), label = as.matrix(test[, 5]))
valids <- list(test = dtest)
# Defining the objective function for a multi-class problem.
params <- list(objective = "multiclass", metric = "multi_error", num_class = 3)

# Training lightgbm model.
lightgbm_model <- lgb.train(params, dtrain,100, valids, min_data = 1,learning_rate = 1.0
    ,early_stopping_rounds = 10)

# Lightgbm predicts all probabilities for the 3 classes; use argmax() to get the classified class.
lightgbm_predictions <- predict(lightgbm_model, as.matrix(test[, 1:4]), reshape = TRUE)
```

Model Explainability

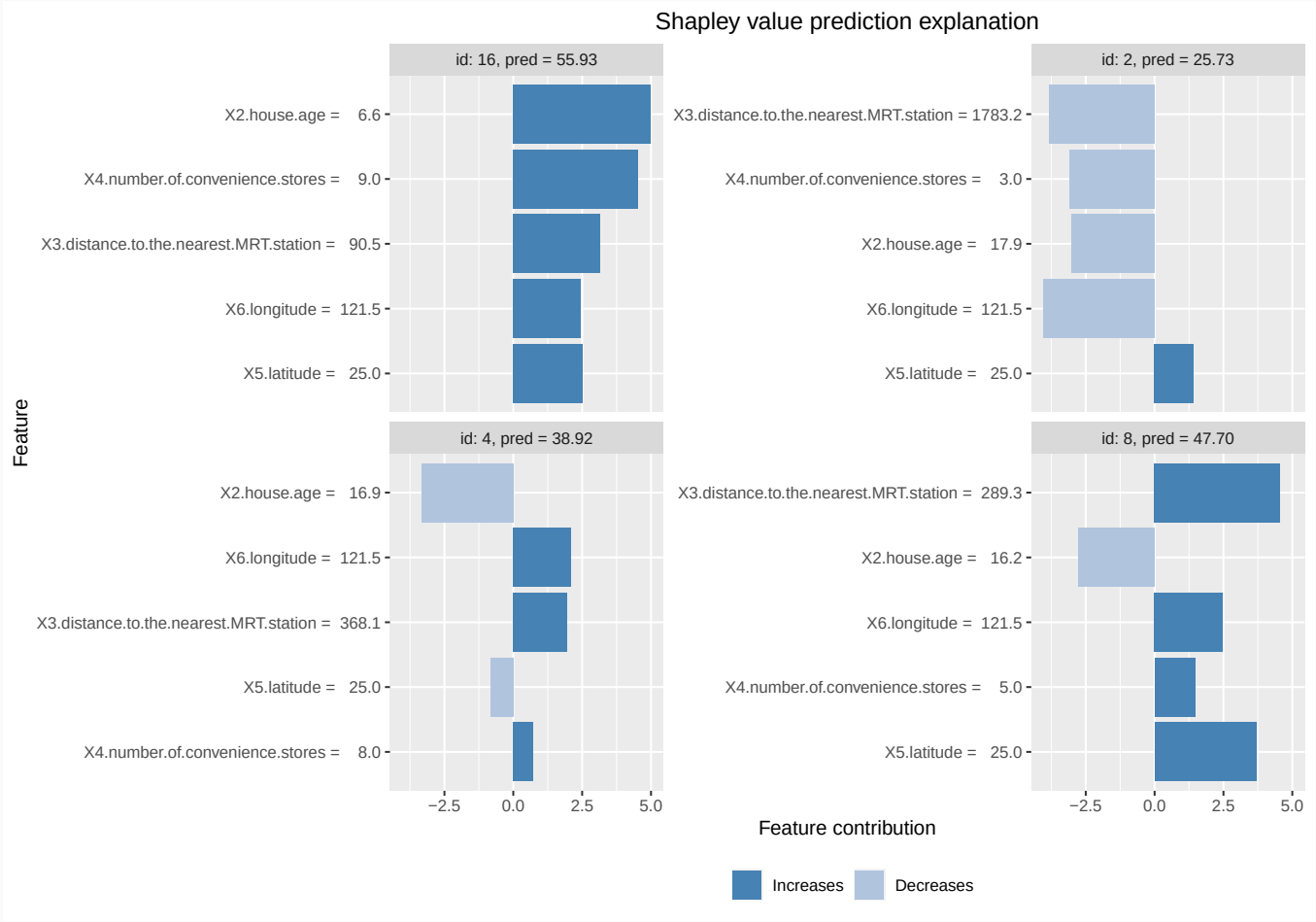
- Majority of the machine learning models are black-box.
- Interpreting and explaining model predictions.
- Explainable machine learning (GDPR law: Right explanation)
    - Global and Local Interpretable models
    - **LIME** (Local Interpretable Model-Agnostic Explanations): `{limer}`
    - **SHAP** (SHapley Additive exPlanations): `{shapr}`
    - **LORE** (Rule-based Explanations)
    - **Anchor**

Model Explainability using {shapr}

```
library(xgboost)
library(shapr)
set.seed(1234)

# We are explaining the prediction of household prices.
df_households <- read.csv("data/realestate.csv")
df_households_filtered <- df_households[, c(-1,-2)]
colnames(df_households_filtered)[6] <- "price"

data_split <- initial_split(df_households_filtered, prop = .7)
train <- training(data_split)
test  <- testing(data_split)

features <- setdiff(names(train), "price")
train_features <- as.matrix(train[, features])
train_class <- as.matrix(train$price)
test_features <- as.matrix(test[, features])

# Fitting a basic xgboost model to the training data
xgboost_model <- xgboost(data=train_features, label=train_class, nround=10, verbose = FALSE)

# Prepare the data for explanation
model_explainer <- shapr(train_features, xgboost_model)
# Expected value without the prediction.
expected_price <- mean(train_class)
```

Model Explainability (2)

```
explanation <- explain(test_features, approach = "empirical",
                       explainer = model_explainer, prediction_zero = expected_price)

# Plot the resulting explanations for observations 1 and 6
plot(explanation, plot_phi0 = FALSE, index_x_test = c(2, 4, 8, 16))
```



References

[1]