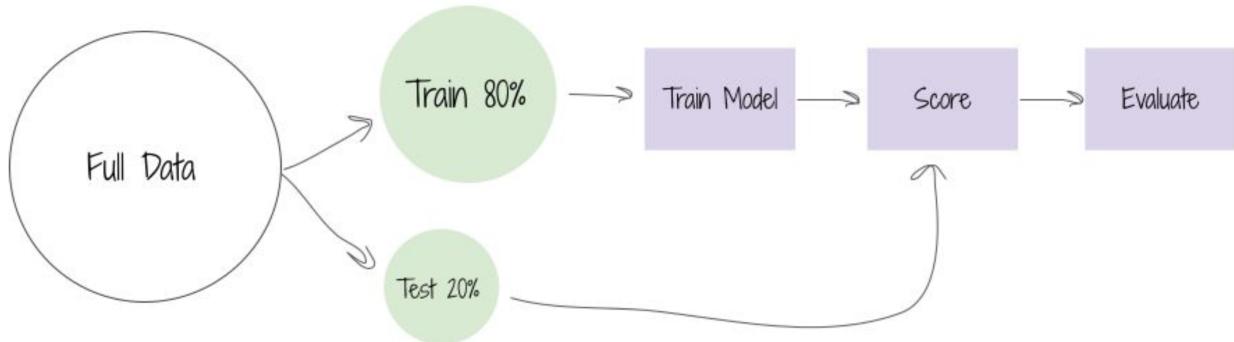


Essential ML for data analyst

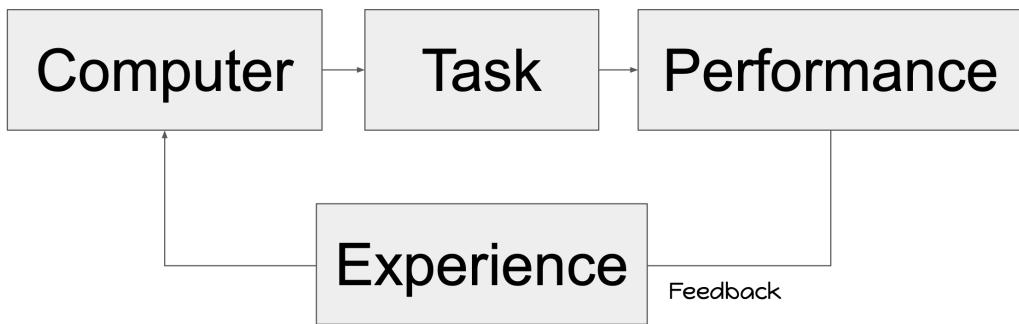
Simple ML pipeline



โปรแกรมการเทรนโมเดลที่ง่ายที่สุดคือ Train Test Split

Essential ML

- What exactly is Machine Learning คือ **The ability to learn without being explicitly programmed. Computer learn from data.**



Computer จะเรียนรู้ผ่านประสบการณ์และ Feedback ทำให้มันเก่งขึ้นเรื่อยๆ

data set → คือข้อมูลที่อยู่ในตาราง

data point → row

Features → คือตัวแปรต้นที่ใช้เทรนโมลเดล

Label/Target → คือค่าลับน์ที่เราจะนำมายหรือตัวเปรียบ

- ศัพท์ใหม่จาก **algorithm linear regression**

- สมการคือ $y = b_0 + b_1*x$ โดย model จะหาค่า b_0 b_1 (**weight**) ให้ได้ error น้อยที่สุด โดยการปรับค่าที่ 0.001 (**Learning rate**) point ถ้า error ลดลงแสดงว่ามาถูกทาง แต่ถ้าปรับแล้ว error เพิ่มก็จะทำการทำงานเรียกเทคนิคแบบนี้ว่า **convergence**

- Supervised Learning VS. Unsupervised learning

Supervised Learning → คือการนำ Features ไปช่วยในการนำมายตัวเปรียบ

Unsupervised learning → คือการใช้เดต้าเพื่อหา pattern หรือการทำ segmentation ที่นักหนังสือนำมาย

- clustering
- association
- dimension reduction

Reinforcement Learning → ใกล้เคียงกับ Ai มากที่สุด โดย **agent** คือ computer ที่เรา (**environment**) จะสอน การสอนคือ **action** ถ้าทำถูกเราจะให้ **reward** ถ้าผิดเราจะได้ **punishment**

penalty และคอย **observation** ตลอดเวลา โดยในมุมของ agent จะพยายาม maximize rewards

Supervised Learning	Unsupervised Learning
Has features (x) and labels (y)	Has features (x) without labels (y)
The goal is PREDICT	The goal is to SUMMARISE
Example algorithms <ul style="list-style-type: none">- Regression- Classification	Example algorithms <ul style="list-style-type: none">- Clustering- Association Rules- Principal Component Analysis



คอร์สเราไฟฟ้าสก็ท supervised learning



ประเด็นสำคัญคือการเลือก features เพื่อให้เหมาะสมกับการทำนายตัวแปรตาม เพราะคอมพิวเตอร์ไม่สามารถเข้าใจชื่อคลิปนี้เราได้ เช่น Bank → รายได้ หนี้สิน จ่ายเงินตรงเวลาใหม่ จำนวนบัตรเครดิต และอื่นๆ

และถ้าเราดูแค่ Supervised Learning จะมีสองแบบคือ

1. **Regression** คือการทำนายเป็นตัวเลข numeric
2. **Classification** คือการทำนายเป็น categorical

1. Regression	2. Classification
Predict numeric labels	Predict categorical labels
Examples <ul style="list-style-type: none"> - house price - customer satisfaction - personal income - how much a customer will spend 	Examples <ul style="list-style-type: none"> - yes/ no question - churn prediction - conversion - weather forecast - default prediction
100, 200, 250, 190, 300, 500, etc.	0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, etc.

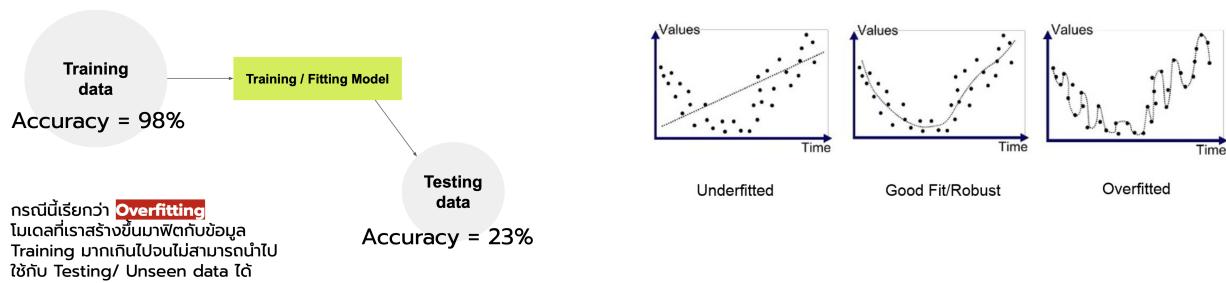
- **3 Steps to build ML** → Prepare data, Train Algorithm, Test/evaluate algorithm

- **Train test split** → คือการแบ่งข้อมูลออกเป็น 80:20 หรือสัดส่วนอื่นๆ ขึ้นกับ full data ที่เราเชื่อว่าใหญ่แค่ไหน

Training set → คือชุดข้อมูลที่เราใช้ในการสร้าง model **เหมือนตอนเราเรียนในห้อง**

Test set → คือชุดข้อมูลที่เรานำมาไปเก็บกับผลของ model ที่ออกแบบ **เหมือนตอนเราสอบ final exam**

Overfitting → คือการที่ model มัน fit เข้ากับข้อมูลที่เทรนมาก ๆ แต่ไม่สามารถเอาไปใช้งานได้จริงกับข้อมูลใหม่ หรือ Under fitting คือ model ไม่ fit กับข้อมูลที่เทรน **การดูให้ดีที่ Train Accuracy เทียบกับ Test Accuracy ถ้า Train accuracy > Test accuracy เยอะๆ 60% - 80% แสดงว่าเราเจอปัญหา Over Fitting**

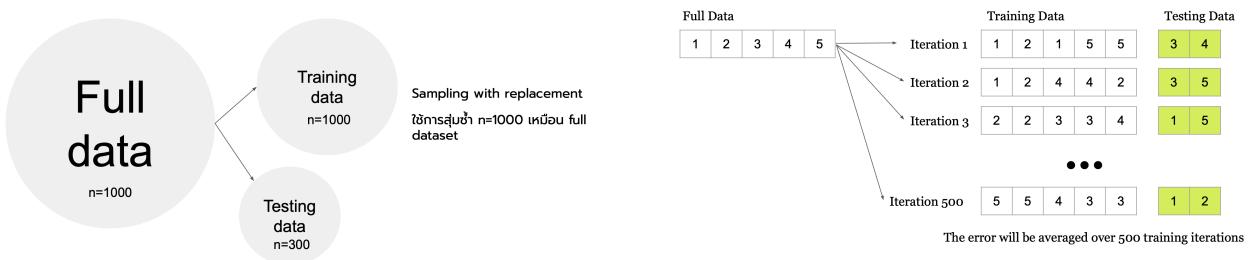


- Train Test split อาจจะไม่ได้ใช้วิธีที่ดีที่สุด เพราะเราเห็น errors แค่ครั้งเดียว สุ่มครั้งเดียว นี่คือข้อจำกัดของมัน เราจะมีเกณฑ์ที่เรียกว่า **Resampling** คือการทำซ้ำ ที่เจ็งกว่าในการวัดค่า errors ได้ดีขึ้น

- Leave one out CV คือ การ แทนตามจำนวน k ของ Full data โดยเก็บ Test 1 ตัวดังรูปขวามือ โดยการวัดผล errors จะเก็บทุกรอบและมาเฉลี่ยตอนรอบสุดท้าย



- Bootstrap คือการที่เราสุ่ม Train สมบูต 70% สุ่ม Test 30% $n=1000$ เพราะฉะนั้น Train data จะต้องมี 700 ปะ แต่ไม่ใช่ train data จะสร้างโดยการสุ่มแล้วขึ้นมาเพื่อให้มันเท่ากับ Full data ดังรูป และทำการสุ่ม defult อยู่ที่ 25 ครั้ง ประยุกต์เวลา กว่า วิธีแรกและประสึกภาพดี



- K-Fold cross validation คือการที่แบ่งข้อมูลออกตามจำนวน k ที่กำหนด โดยก็จะไปเท่ากับ 5 หรือ 10 เช่นนี้ $k=1000$ $k=5$ เพราะฉะนั้นแต่ละ fold จะมีข้อมูล 200 แสดงดังรูป จากนั้นก็รันที่ ละรอบและเก็บค่า errors ไว้และมาเฉลี่ยตอนสุดท้าย วิธีนี้เหมาะสมกับ classical ML

1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5
1	2	3	4	5

iteration 1: train {2,3,4,5} test {1} -> error 18%

iteration 2: train {1,3,4,5} test {2} -> error 20%

iteration 3: train {1,2,4,5} test {3} -> error 30%

iteration 4: train {1,2,3,5} test {4} -> error 15%

iteration 5: train {1,2,3,4} test {5} -> error 19%

Average error = $(18+20+30+15+19) / 5 = 20.4\%$

ปกติเราเน้นมันใช้ค่า K=5 หรือ K=10

- **R Studio**

ติดตั้ง library → tidyverse, caret

1. Split data
2. Train model
3. Score model
4. Evaluate model

```
library(tidyverse)
library(caret)

## 1. split data

train_test_split <- function(data, size=0.8) {

  set.seed(42) ## ล็อกผลลัพธ์ไว้ต่อ

  n <- nrow(data) ## กำหนดแຄวเพื่อใช้ในการสุ่ม
  train_id <- sample(1:n, size*n) ## เราสุ่มเดต้าออกมา 80% สำหรับเทรน
  train_df <- data[train_id, ] ## นำเอา train_id ตະกິບາຫັນເຫດເພື່ອ train
  test_df <- data[-train_id, ] # ข้อมูลໃຫຍ່ເກີດຈາກ train df ກິນາມາ
```

```

        return(list(train_df, test_df)) ## เก็บค่าที่จะส่งออกได้ก็ง Train df
    }

prep_df <- train_test_split(mtcars, size=0.8) ## ลองเรียกใช้ function

## 2. train model
ctrl <- trainControl(method = "boot", ## "LOOCV", "cv"
                      number = 25) ##การเปลี่ยน resampling

model <- train(mpg ~ hp + wt + am, ## y=x1+x2...
                data = prep_df[[1]],
                method = "lm",
                trControl = ctrl) ##การเปลี่ยน resampling

```

```

> model
Linear Regression

25 samples
 3 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 25, 25, 25, 25, 25, 25, ...
Resampling results:

RMSE      Rsquared     MAE
2.771533  0.8345874  2.189929

Tuning parameter 'intercept' was held constant at a value of TRUE
>

```

การวัดผลเราจะดู RMSE , R squared, MAE

```

## 3. score model คือเอา model ที่สร้างมาไป predict test data

pred_mpg <- predict(model, newdata = prep_df[[2]])

```

```

## 4. evaluate model

actual_mpg <- prep_df[[2]]$mpg

## error = actual - prediction
test_mae <- mean(abs(pred_mpg - actual_mpg))
test_rmse <- sqrt(mean((pred_mpg - actual_mpg)**2))

print(test_mae)
print(test_rmse)

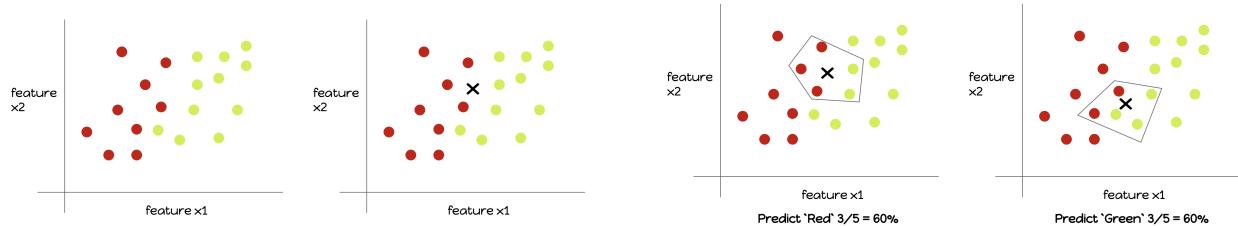
```

Homework

- K-Nearest Neighbors
- Find the best parameter
- Normalization
- Diabetes case study

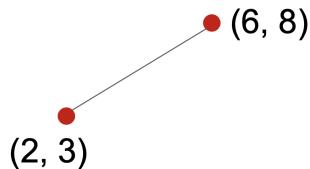
K-Nearest Neighbors

គោរកទំនាក់ទំនងនៃការសម្រេចបន្ថែម



สิ่งที่เราต้องคำนึงคือการสุ่มเลือกจุดที่ใกล้บันมาก่อนจะหาทางและจุดที่สุ่มต้องเป็นเลขคี่เพื่อกำหนดว่าจะมีผู้แพ้ผู้ชนะฝ่ายการโหวตเลือก

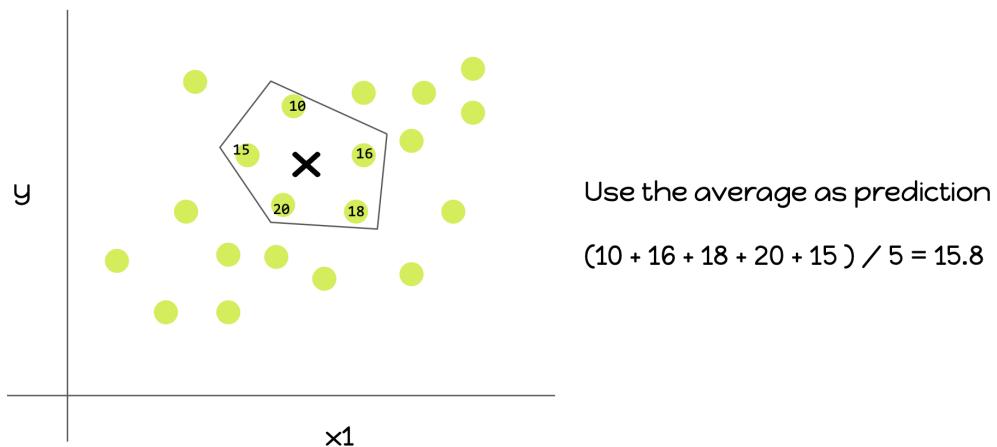
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



สูตรของ Euclidean ในการหาระยะทาง สามารถคำนวณใน r ได้ง่าย

```
point_1 <- c(2,3)
point_2 <- c(6,8)
d <- sqrt( (2-6)**2 + (3-8)**2 )
print(d)
```

หรือเราสามารถหาค่าเฉลี่ยเพื่อแก้ปัญหา regression ได้



Find the best parameter

```
library(tidyverse)
library(caret)
library(mlbench) ##download data set

data() ##ดูว่ามีเดต้าอะไรใน MLbench บ้าง

data("BostonHousing") ##การโหลดเดต้าเข้ามา
data("PimaIndiansDiabetes")

## clean data before train
clean_df <- mtcars %>%
  select(mpg, hp, wt, am) %>% ## คำสั่งนี้เราจะเลือกคอลัมน์ที่มีผลกับตัวแปรตาม
## imputation เราจะแทนที่ missing by mean หาโดย mean(mtcars$hp) ใน
  mutate(hp = replace_na(hp, 146.68),
         wt = replace_na(wt, 3.21)) %>%
  drop_na() ## เราจะลบ missing ทั้งหมดถ้าเจอ NA < 3-5
```

```
## linear regression
lm_model <- train(
  mpg ~ ., ## ใช้ . แทนตัวแปรตั้งที่เราเลือกจากด้านบน เพราะมันรู้อยู่แล้วว่ามีอะไรบ้าง
  data = clean_df,
  method = "lm"
)

## KNN
knn_model <- train(
  mpg ~ ., ## ใช้ . แทนตัวแปรตั้งที่เราเลือกจากด้านบน เพราะมันรู้อยู่แล้วว่ามีอะไรบ้าง
  data = clean_df,
  method = "knn"
)
```

เราได้เทรนเดต้าออกเป็นสองโมเดลคือ linear regression และ Knn ซึ่งผลลัพธ์แสดงดังรูป

```

> lm_model
Linear Regression

32 samples
3 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 32, 32, 32, 32, 32, 32, ...
Resampling results:

  RMSE     Rsquared     MAE
2.807091  0.8295991  2.279069

Tuning parameter 'intercept' was
held constant at a value of TRUE

> knn_model
k-Nearest Neighbors

32 samples
3 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 32, 32, 32, 32, 32, 32, ...
Resampling results across tuning parameters:

      k    RMSE     Rsquared     MAE
      5   3.477360  0.7126952  2.640049
      7   3.655322  0.7014038  2.758610
      9   3.875137  0.6808807  2.871619

RMSE was used to select the optimal model using the
smallest value.
The final value used for the model was k = 5.

```

คำความของ knn model คือ เราจะเลือก k ที่เท่าไหร่ คำตอบคือเราใช้ matrix ตัวไหนวัด เช่น ถ้าใช้ RMSE ค่ากี่น้อยกี่สุดคือ 3.47736 เลือก k เท่ากับ 5 (**สังเกตบรรทัดสุดท้าย**)

เพรอนั้นเราสามารถเลือกได้ว่าจะใช้ matrix ตัวไหนในการเลือก k (ส่วนมากก็ RMSE ที่ใช้เลือก)

```

## KNN
set.seed(42) ##เซตให้ผลลัพธ์เราเหมือนเดิม

(knn_model <- train(
  mpg ~ ., ## ใช้ . แทนตัวแปรตันที่เราเลือกจากด้านบนเพระมันรู้อยู่แล้วว่ามีอะไรบ้าง
  data = clean_df,
  method = "knn",
  metric = "MAE" ##การเปลี่ยนว่าจะใช้ metric ในเลือก k
))

```

*สังเกตว่างเส้นกรอบกั้งหมดของโนําเดลคือเหมือนกันกับสั้นรันมันจะขึ้นผลลัพธ์ตรง console ให้เลย

สังเกตว่า performance ของ model เนี่ยจะขึ้นกับ parameter ที่เราเลือกและการจะเลือกก็ต้องมาดูว่าเราจะใช้ metric ใน จำนวนแควร์เป็นสิ่งสำคัญ กั้งจำนวนคอลัมน์ด้วย ให้เลือกเจ้ากี่ เกี่ยวข้องกับ y ถ้าไม่เกี่ยวข้อง performance จะลดลง

การเซต resampling เป็น cv

```

ctrl <- trainControl(
  method = "cv",
  number = 5, ##fold
  verboseIter = TRUE ##โชว์ progress ว่าอยู่ fold ไหนแล้ว
)

(knn_model <- train(
  mpg ~ ., ## ใช้ . แทนตัวแปรตั้งที่เราเลือกจากด้านบน เพราะมันรู้อยู่แล้วว่ามีอะไรบ้าง
  data = clean_df,
  method = "knn",
  metric = "MAE", ##การเปลี่ยนว่าจะใช้ metric ไหนเลือก k
  trControl = ctrl
))

```

เลือกค่า K เองละ ให้ set data.frame และเลือก ค่า k ที่เราต้องการ เสร็จแล้วนำไปใส่ใน model ด้วย `tuneGrid` และถ้าเราไม่รู้ว่าจะใช้อะไรดี เราใช้ `tuneLength = 2` คือสุ่ม K มาสองตัวแต่เป็นอะไรไม่รู้

```

## KNN
set.seed(42)

## grid search
k_grid <- data.frame(k = c(3,5))

ctrl <- trainControl(
  method = "cv",
  number = 5, ##fold
  verboseIter = TRUE ##โชว์ progress ว่าอยู่ fold ไหนแล้ว
)

(knn_model <- train(
  mpg ~ ., ## ใช้ . แทนตัวแปรตั้งที่เราเลือกจากด้านบน เพราะมันรู้อยู่แล้วว่ามีอะไรบ้าง
  data = clean_df,
  method = "knn",

```

```
metric = "MAE", ##การเปลี่ยนว่าจะใช้ metric ไหนเลือก k  
trControl = ctrl,  
tuneGrid = k_grid ## ถ้าเป็น tuneLength คือมันจะสุ่มค่าของ k ให้ตามจำนวน  
))
```

***k** ยิ่งตัว มีโอกาสเกิด overfit k มาก มีโอกาส underfit

การแทน cv ชี้ บันทึกวันที่ เราช��นรอน ตามจำนวน number และวนรอบใหม่ ตาม repeats
บันคือ **repeatedcv** บันจะวัดได้ແມ່ນຢໍາເຊັນ

```
ctrl <- trainControl(  
  method = "repeatedcv",  
  number = 3, ##fold  
  repeats = 5,  
  verboseIter = TRUE ##ໃຊ້ progress ວ່າວຍໆ fold ໄහນແລ້ວ  
)
```

Save Model และ Read Model

```
## save model  
## .RDS extension  
saveRDS(knn_model, "knnModel.RDS")  
  
## read model  
friend_laptop <- readRDS("knnModel.RDS")
```

Normalization

คือการทำให้ตัวแปลงตัวต่างๆ มีสเกลหรือช่วงของข้อมูลที่เท่ากัน ควรทำทุกครั้งก่อนสร้าง model มีสองวิธี 1. min-max และ 2. standardization

```

## Normalization
## 1. min-max (feature scaling 0-1)

x <- c(5, 10, 12, 15, 20)

minmaxNorm <- function(x) {
  (x-min(x)) / (max(x)-min(x))
}

## 2. standardization -3, +3
## center and scale

zNorm <- function(x) {
  (x-mean(x)) / sd(x)
}

```

การทำ Normalization ควรทำที่ Train data เพราะถ้าเราทำที่ Full data อาจจะมีข้อมูลบางส่วนหลุดหายัง Test เช่นค่าเฉลี่ย เมื่อونกับการเห็นข้อสอบก่อนเริ่มทำ เพราะฉนั้นกระบวนการที่แท้จริงคือ ทำ Normalization ที่ Train data และ จะได้ X bar, SD เพื่อใช้ scaling กับ Test data ซึ่งมีวิธีที่ง่ายกว่าการคำนวณด้วยมือด้านบน คือ `preProcess` ตามสูตรด้านล่าง

```

## Pre-process
train_df <- mtcars[1:20, ]
test_df <- mtcars[21:32, ]

##เลือกเอาแบบไหนแบบหนึ่ง
## 1.min-max
transformer <- preprocess(train_df,
                           method = c("range"))

## 2.compute x_bar, x_sd
transformer <- preprocess(train_df,
                           method = c("center", "scale")) ##หาค่าเฉลี่ย

```

```
train_df_z <- predict(tranformer, train_df)
test_df_z <- predict(tranformer, test_df) ##สังเกตนะว่าเราใช้ x, sd และ
```

Diabetes case study

```
## build ML to classify diabetes patients
## binary classification

library(tidyverse)
library(caret)
library(mlbench)
library(MLmetrics)

## loading dataset
data("PimaIndiansDiabetes")
df <- PimaIndiansDiabetes

## explore data
glimpse(df) ##ดูข้อมูล คอลัมน์

mean(complete.cases(df)) ##หา missing value ถ้าเป็น 1 แสดงว่าข้อมูล ok
## พิมพ์ complete.cases(df[1:10, ]) ใน console เพื่อดูว่าแต่ละคอลัมน์มี Missing value หรือไม่

## select variables
df_starter <- df %>%
  select(2,5,6,8,diabetes) ##ใช้ ตบ. หรือชื่อคอลัมน์ก็ได้

## 1. split data
set.seed(42)
n <- nrow(df_starter)
id <- sample(1:n, size = 0.8*n)
train_df = df_starter[id, ]
test_df = df_starter[-id, ]
```

```

## 2. train
set.seed(42)

ctrl <- trainControl(method = "cv",
                      number = 5,
                      ## pr = precision + recall ต้องเพิ่ม metric ให้
                      ## สองบรรกัดด้านล่างเขียนเพื่อสามารถก่อจุบค่าอื่นๆ เช่น Pr
                      summaryFunction = prSummary,
                      classProbs = TRUE)

logis_model <- train(diabetes ~ .,
                      data = train_df,
                      method = "glm",
                      ## เราจะเปลี่ยนให้แสดงผลแบบไหนในการจูนค่า เช่นเราจะใช้ R
                      metric = "AUC",
                      trControl = ctrl)

## 3. score
p <- predict(logis_model, newdata = test_df)

## 4. evaluate
mean(p == test_df$diabetes)

## confusion matrix
table(test_df$diabetes, p, dnn=c("actual", "pred")) ##เพื่อดูตาราง confusion matrix

##สามารถใช้ฟังก์ชันนี้คำนวณได้เลย
confusionMatrix(p, test_df$diabetes,
                 positive="pos", ## คลาสที่เราสนใจ
                 mode="prec_recall") ## เปลี่ยนโหมดเป็น Precision และ Recall

## เราจะดู Accuracy, recall , precision, F1 เลือกเอาตามบาร์บุก

```

การเลือกตัวแปร ที่คิดว่าเกี่ยวข้องกับตัวแปรตาม ง่ายๆคือดูจากค่าเฉลี่ยของแต่ละ class ใน ตัวแปรตามถ้ามันต่างกันมากและเป็นไปตามความจริงแสดงว่าคอลัมน์นั้นมีผล

```
##พิมพ์ใน console

df_starter %>%
+   group_by(diabetes) %>%
+   summarise(mean(age), mean(mass))
```

```
Rows: 768
Columns: 5
$ glucose <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 1...
$ insulin <dbl> 0, 0, 0, 94, 168, 0, 88, 0, 543, 0, 0, ...
$ mass     <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31...
$ age      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54...
$ diabetes <fct> pos, neg, pos, neg, pos, neg, pos, neg...
> df_starter %>%
+   group_by(diabetes) %>%
+   summarise(mean(age), mean(mass))
# A tibble: 2 × 3
  diabetes `mean(age)` `mean(mass)`
  <fct>        <dbl>       <dbl>
1 neg            31.2        30.3
2 pos            37.1        35.1
> |
```

จะเห็นว่าเมื่ออายุหรือ นน มากขึ้น class ที่ได้คือเป็นเบาหวานมัน make sense นะ แล้วค่าเฉลี่ยจะต่างกันมากแสดงว่าคอลัมน์นี้มีผล

การหา Baseline Accuracy คือความแม่นยำจากการที่เราไม่ใช้ model ใช้เพื่อเปรียบเทียบกับความแม่นยำที่ได้จาก model

การหาคือการเอา จำนวนแควรของ class ที่เรสนใจ เช่น class เป็นโรคเบาหวาน หารกับจำนวนแควรทั้งหมด

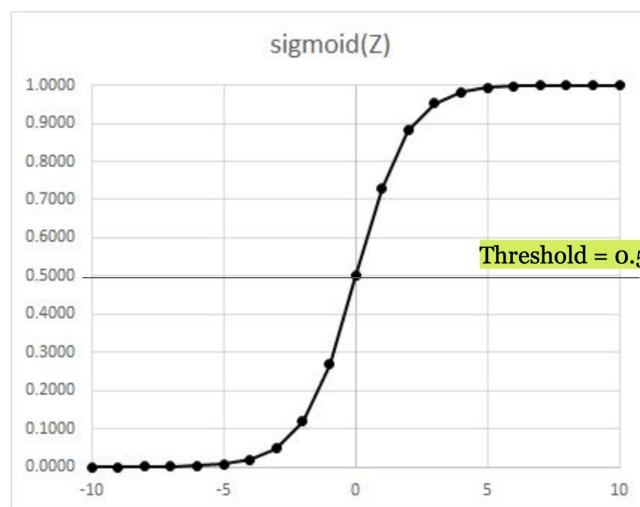
ข้อควรระวังการใช้แค่ accuracy อย่างเดียวคงจะยังไม่แม่นพอ เช่นถ้าเราเอาคลาสที่บกวนไทย เป็นโควิด 1 ใน 1000 คนแสดงว่ามีความแม่นยำอยู่ 0.1 % แต่ถ้าเราเอาความแม่นยำที่คิดจากคนไม่เป็นโควิด 999/1000 เท่ากับ 99.9 % จะเห็นว่าแม่นเกินไปนะ เราเรียกแบบนี้ว่า **Imbalance** ข้อมูลที่ต้อง 50:50

เพราะฉะนั้น เราจะมี Confusion Metric กี่ช่วยเพิ่มในการตัดสินใจ เช่น AUC คือกราฟที่พล็อตระหว่าง recall และ precision ยิ่งเส้นมันจะเต็มกราฟมากเท่าไหร่หรือใกล้ 1 ยิ่งดี , Recall, Precision, F1

- Classification vs. Regression

Classification vs Regression (Review)

- classification
 - การ set threshold ใน Logistic ขึ้นอยู่กับระบบกว่าเราจะให้มันเพิ่มหรือลดแต่ส่วนใหญ่จะ set อยู่ที่ 0.5 สมมุติว่าจากราฟสูงสุดคือ 1 คือทำนายเป็น Yes และด้านล่างกราฟคือ 0 คือทำนายเป็น No ในช่วงโควิดคุณหมออวยากให้ Model ทำนายเป็น Yes หรือ No ก็ต้องอยากรู้ว่าทำนายเป็น Yes ถูกปะ เพราะจะได้กวาดคนที่เป็นโควิดมารักษา เพราะฉะนั้นการ set threshold จะต้อง set ต่ำๆ เช่น 0.2



Decision Tree

- คือการถามคำถาม Yes No และเลี้ยงไปทางนั้น
- ข้อดีคือรันง่าย แต่ข้อเสียคือเกิด Overfit ได้ง่าย ผลลัพธ์ใกล้เคียงกับตระกูล Regression แต่เทrnanya กว่า

อย่าลืม split data ก่อนหน้า

```
library(tidyverse)
library(caret)
library(mlbench)

## Load data
data("PimaIndiansDiabetes")
df <- PimaIndiansDiabetes

## Clean data
mean(complete.cases(df)) # เลข 1 ดาต้าสมบูรณ์

## Train Model rpart
## Recursive partitioning (decision tree)
ctrl <- trainControl(
  method <- "cv" ,
  number = 5,
  verboseIter = TRUE, # show result in console
  classProbs = TRUE, # we can change threshold 0.5
  summaryFunction = twoClassSummary
)

tree_model <- train(
  diabetes ~ glucose + pressure + insulin + mass + age,
  data = df,
  method = "rpart",
  metric = "ROC", ## เมื่อ Set classProbs ด้านบนเราต้องเปลี่ยนวิธีการวัดผลเป็น
  trControl = ctrl
)
```

```

## Prediction
predict(tree_model, df, type = "prob")[1:10, ]
#คือเมื่อเรา set Classprobs and summaryFunction เราสามารถเปลี่ยน type และ

## change threshold
probs <- predict(tree_model, df, type = "prob")
p_class <- ifelse(probs$pos >= 0.5, "pos", "neg")
table(df$diabetes, p_class) # สร้าง confusion matrix โดย threshold
#สามารถเปลี่ยน threshold ได้และเอาไปใช้ตามบริบท

```

cp : complexity parameter ถ้าค่าสูงขึ้นแสดงว่า gen ดี **overfitting** จะน้อยลง ส่วน Accuracy น้อยลงตาม

Random Forest

- I like the Most
- คือมันจะสร้างต้นไม้มาเรื่อยๆ ต้นเลยและใช้การโหวต (majority vote) ในการเลือก ทำให้มันเกิดมากและลด **overfitting**
- ซึ่งมันจะมีหลักการของมันอยู่ คือ bagging \leftarrow mtry + hyperparameter โดย mtry คือการสุ่ม จบ. คอลัมน์มาใช้ในแต่ละต้นไม้ซึ่งแต่ละต้นไม้จะสุ่มออกมาไม่เหมือนกันแต่จบ. จะเก่ากับที่เรา set ไว้ ทำให้ต้นไม้ที่เราสุ่มมาไม่เหมือนกันนั้นเองงง

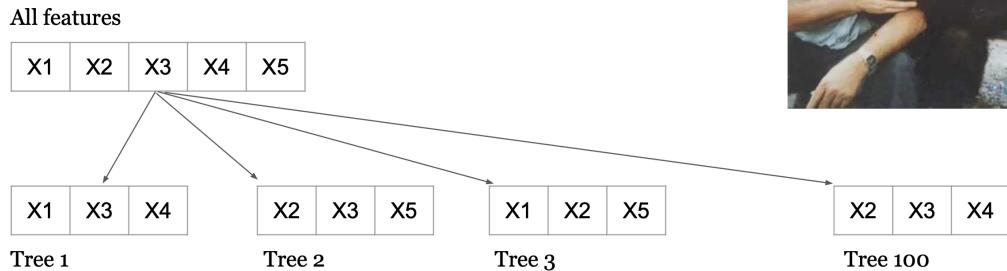


We grow hundreds of
uncorrelated (decision) trees

Combine them to make
prediction (similar to KNN,
majority vote or average)

R Teamwork (Bagging)

Bootstrap + mtry hyperparameter



```
# random forest
ctrl <- trainControl(
  method <- "cv" ,
  number = 5,
  verboseIter = TRUE # show result in console
)

rf_model <- train(
  diabetes ~ glucose + pressure + insulin + mass + age,
  data = df,
  method = "rf", ## ใช้ rf
  metric = "Accuracy",
  tuneGrid = data.frame(mtry = c(2, 3, 4)), # set ค่า mtry สุ่มมา
  trControl = ctrl
)
```

Ridge Regression & Lasso Regression

- Regularization is Reduce Overfitting Technique
- โดยมันจะลดค่า slope ในสมการ regression
- Lasso Regression ใช้ในการเลือก Feature โดยบางครั้งจะตบตัวแปรตัวเป็น 0 เลย ดีกว่า Ridge ที่จะลดค่าแต่ไม่ถึงกับ 0

```

## Ridge Lasso

glmnet_model <- train(
  diabetes ~ glucose + pressure + insulin + mass + age,
  data = df,
  method = "glmnet",
  metric = "Accuracy",
  tuneGrid = expand.grid(
    alpha = 0:1, # 0 is ridge, 1 is lasso ถ้าค่าอยู่ระหว่างนี้เรียกว่า elas
    lambda = c(0.004, 0.04, 0.08)
  ),
  trControl = ctrl
)

#อ่านค่าที่ accuracy ถ้าอันไหนมากก็เลือก alpha กับ lambda ค่านั้น

```

Ensemble Model

- เอาหลาย ๆ model มาช่วยทำนาย

**นำโมเดลหลายๆตัวมาช่วยกัน
ทำนายผล (Majority Vote)**

KNN	Logistic Regression	Ridge Regression	Decision Tree	Random Forest
1	0	0	1	1

Homework