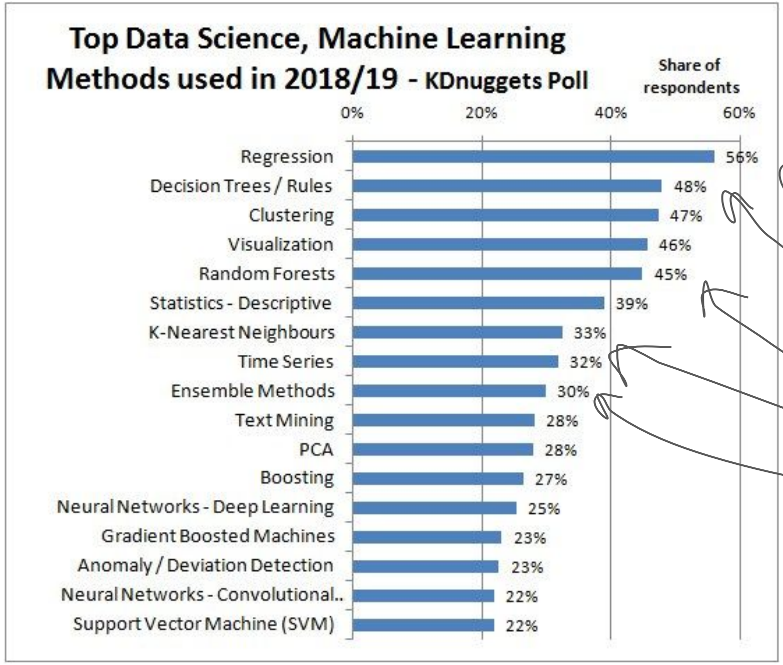
Popularity of Algorithms



Today we'll cover

- Linear Regression
- Logistic Regression
- Regularized Regression
- Decision Tree
- Random Forests (Bagging)
- K-Nearest Neighbours
- Ensemble Model

https://www.kdnuggets.com/2019/04/top-data-science-machine-learning-methods-2018-2019.html

No Free Lunch

No Free Lunch แปลว่า “ไม่มีโมเดลไหนเก่งที่สุด และสามารถตอบโจทย์ได้ทุกปัญหา”

ถ้ามีใครถามว่าโมเดลไหนเก่งที่สุด?
ให้ตอบว่า “It depends” (ขึ้นอยู่กับข้อมูล)
ความท้าทายของ ML คือการหาโมเดลที่ดีที่สุดสำหรับปัญหาที่เรากำลังแก้

Occam’s Razor

Algorithm #1 vs. Algorithm #2

ถ้ามีโมเดลสองตัวที่มี performance ดีเท่าๆกัน ให้เลือกตัวที่
สร้างและอธิบายได้ง่ายกว่า (**choose simpler model**)

How to choose a model

ให้ลองถาม 2 คำถามง่ายๆนี้

1. ปัญหานี้เป็น regression หรือ classification?
2. อยากได้ high accuracy หรือ high interpretability?

- Always choose a simpler model if performances are similar
- Try different algorithms and find the right one.

Caret Package

Learn more at https://topepo.github.io/caret/index.html

Max Kuhn

the author of caret package

Dataset for our projects

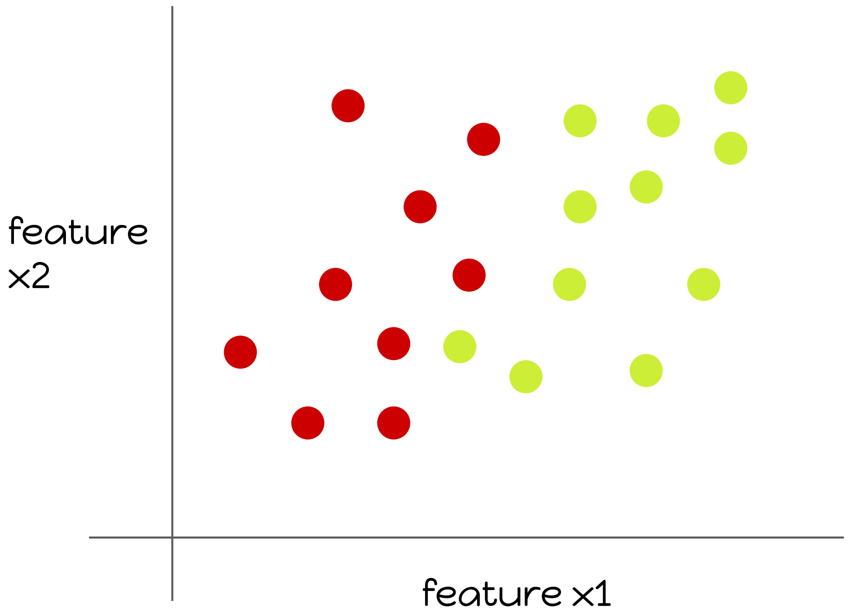
```
## load library
## install.packages(“mlbench”)
library(mlbench)
library(tidyverse)

## load dataset for regression
data(“BostonHousing”)
glimpse(BostonHousing)

## load dataset for classification
data(“PimaIndiansDiabetes”)
glimpse(PimaIndiansDiabetes)
```

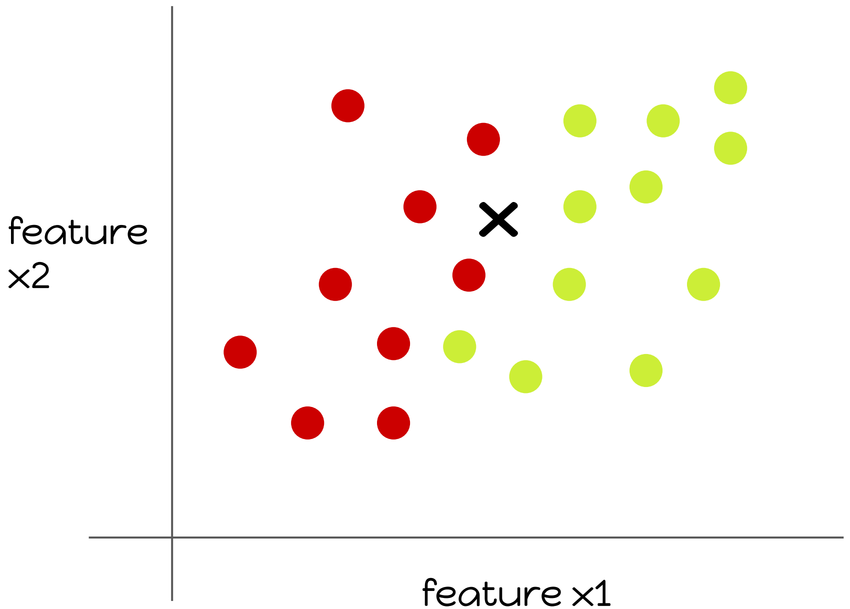
Caret Training Template

```
model <- train(form = y ~ . ,
               data = train_data ,
               method = “lm” )
```

Model that we
want to train

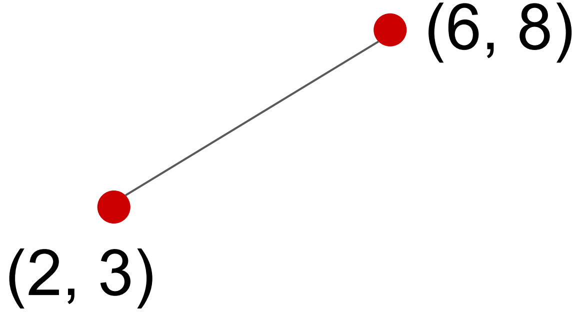
Our first machine





Euclidean Distance

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

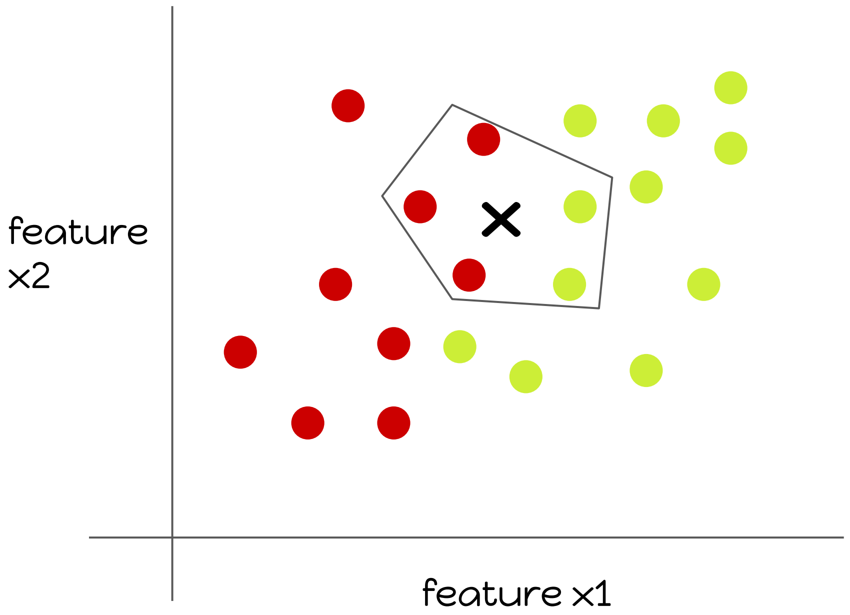


Euclidean Distance

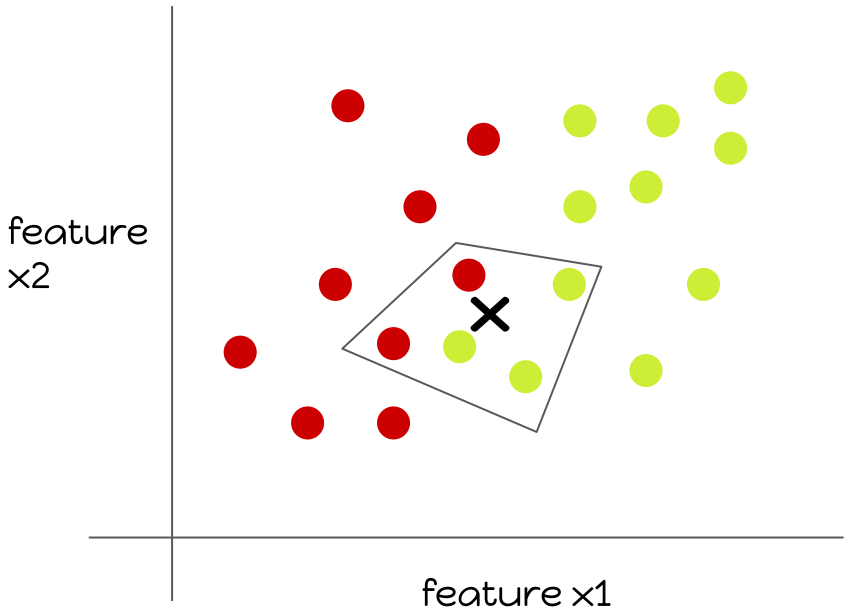
$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

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

We use majority vote to assign label



Predict `Red` 3/5 = 60%



Predict `Green` 3/5 = 60%

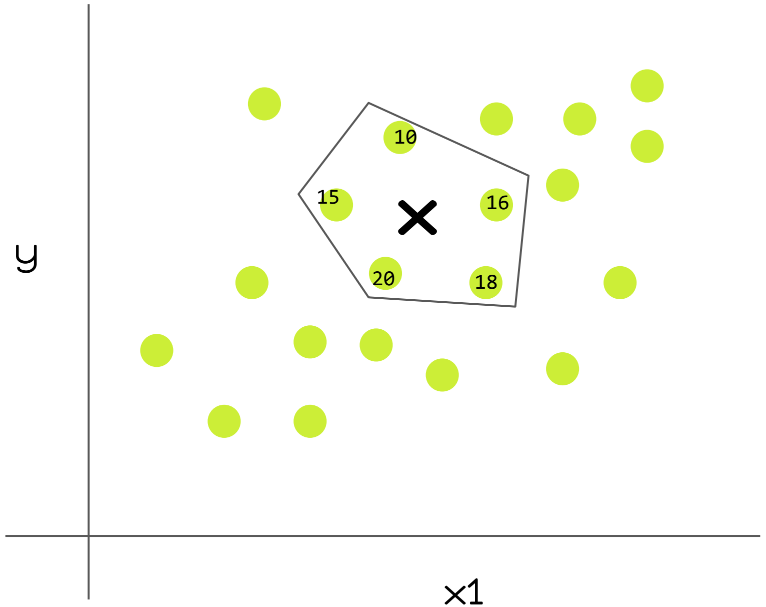
Majority Vote

Steps to train this algorithm

1. แต่งตั้ง สว 250 คน
2. ช่วยกันเลือกนายกฯ

เฮ้ย เด๋วๆๆๆ 55555555555+

We use average value for regression problem



Use the average as prediction

(10 + 16 + 18 + 20 + 15) / 5 = 15.8

K-Nearest Neighbors

1. Choose K
2. Compute distance
3. Majority vote for classification or Average for regression

Train test split (the easiest method)

Prepare dataset first
We'll use split data into training 75% and testing 25%

```
## split data
set.seed(99)
n <- nrow(BostonHousing)
id <- sample(n, size = n*0.75, replace=FALSE)
train_data <- BostonHousing[id, ]
test_data <- BostonHousing[-id, ]
```

Very easy to train a machine in R

```
## train model
set.seed(99)
knn_model <- train(medv ~ .,
                   data = train_data,
                   method = "knn")

## test model
p <- predict(knn_model, newdata = test_data)

## rmse
rmse <- sqrt(mean((p - test_data$medv)**2))
```

KNN + K-Fold CV

```
## train model
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5, verboseIter = TRUE)
knn_model <- train(medv ~ .,
                   data = train_data,
                   method = "knn",
                   trControl = ctrl)

## test model
p <- predict(knn_model, newdata = test_data)

## rmse
rmse <- sqrt(mean((p - test_data$medv)**2))
```

5 Fold Cross Validation

Random Search

```
## train model
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5, verboseIter = TRUE)
knn_model <- train(medv ~ .,
                   data = train_data,
                   tuneLength = 5,
                   method = "knn",
                   trControl = ctrl)

## test model
p <- predict(knn_model, newdata = test_data)

## rmse
rmse <- sqrt(mean((p - test_data$medv)**2))
```

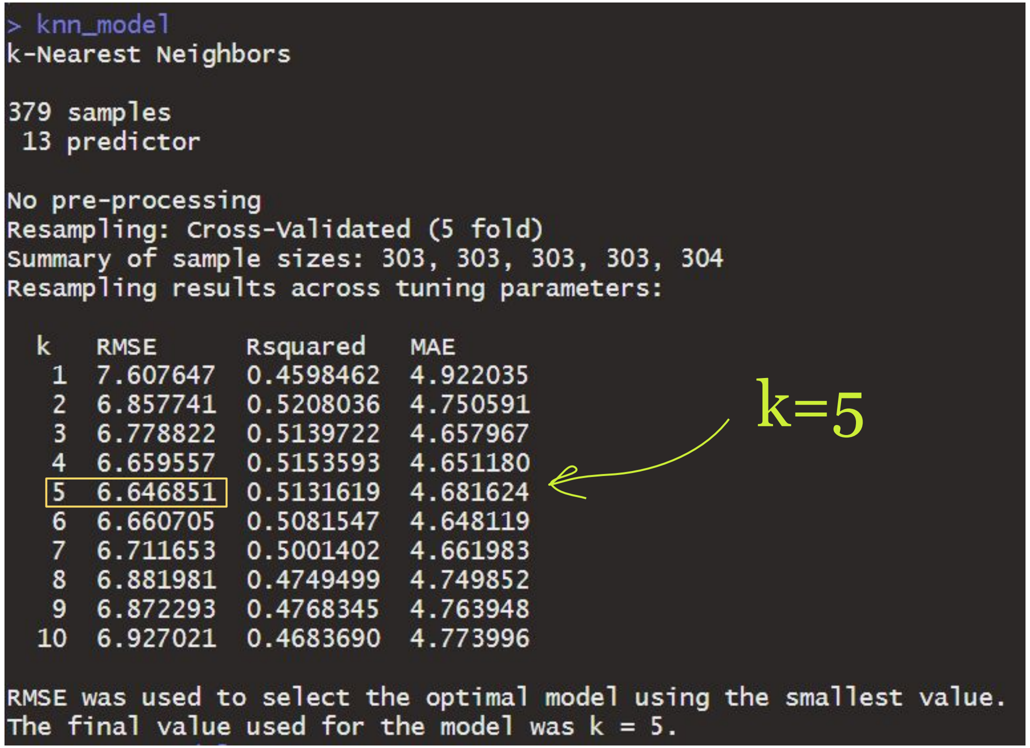
Try 5 values of K

Grid Search

```
## create grid
myGrid <- expand.grid(k = 1:10)

## train model
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5, verboseIter = TRUE)
knn_model <- train(medv ~ .,
                   data = train_data,
                   tuneGrid = myGrid,
                   method = "knn",
                   trControl = ctrl)

## test model
p <- predict(knn_model, newdata = test_data)

## rmse
rmse <- sqrt(mean((p - test_data$medv)**2))
```

The values we select ourselve :D

Grid Search Result

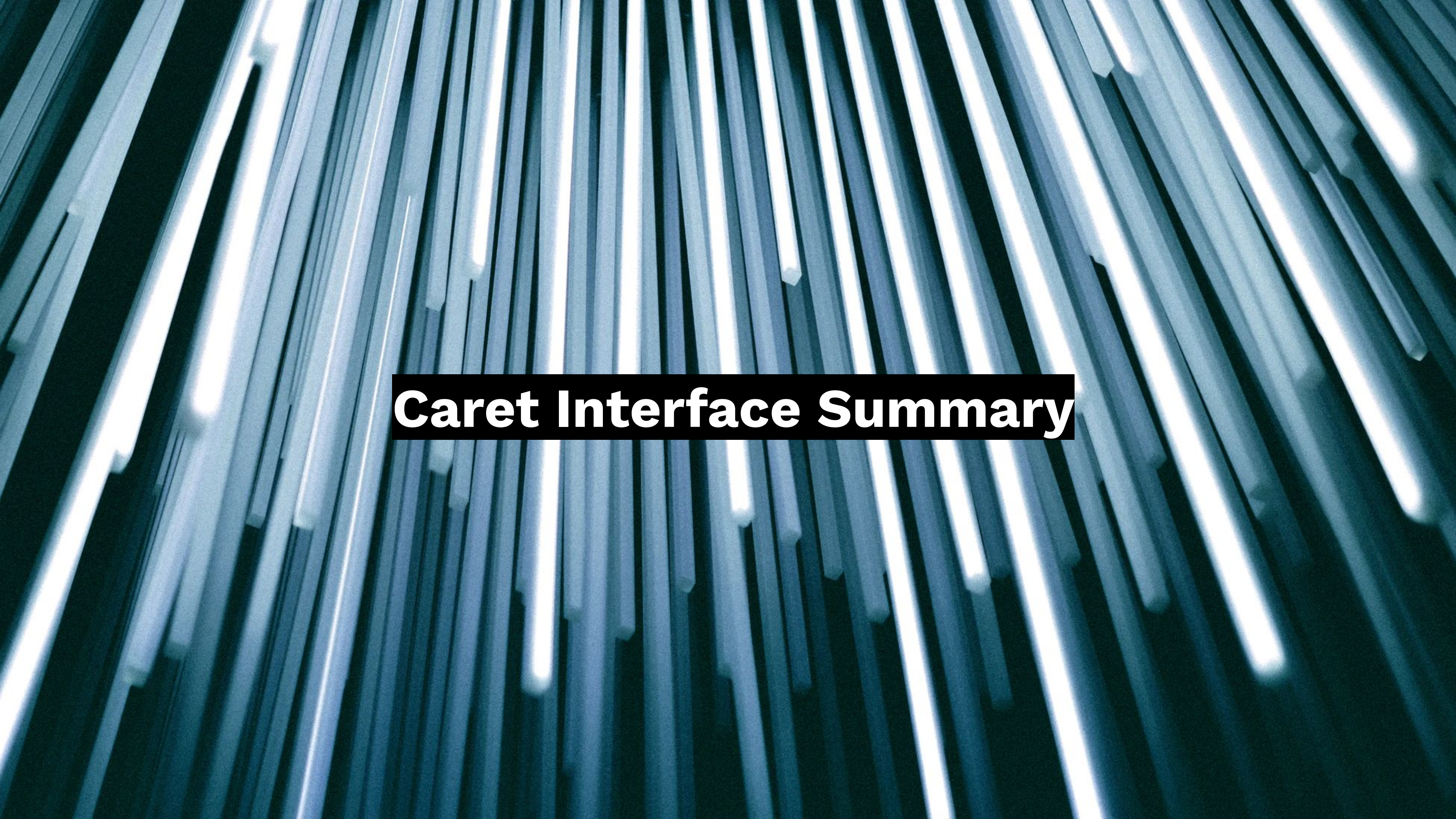


```
> knn_model
k-Nearest Neighbors

379 samples
 13 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 303, 303, 303, 303, 304
Resampling results across tuning parameters:

  k   RMSE      Rsquared   MAE
   1  7.607647  0.4598462  4.922035
   2  6.857741  0.5208036  4.750591
   3  6.778822  0.5139722  4.657967
   4  6.659557  0.5153593  4.651180
   5  6.646851  0.5131619  4.681624
   6  6.660705  0.5081547  4.648119
   7  6.711653  0.5001402  4.661983
   8  6.881981  0.4749499  4.749852
   9  6.872293  0.4768345  4.763948
  10  6.927021  0.4683690  4.773996

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

Cross Validation ช่วยเราเลือกค่า k ที่ทำให้ RMSE ต่ำที่สุด ตอนเรา train model

Key Learning

1. KNN เข้าใจง่ายทำงานได้โอเคร ถ้า feature ไม่เยอะมาก
2. KNN ใช้ได้ทั้ง regression/ classification
3. K ใน KNN คือค่า hyperparameter ที่เราเปลี่ยนได้
4. เราเลือก K ที่ทำให้ train RMSE ต่ำที่สุด
5. train RMSE ต่ำที่สุดไม่ได้แปลว่าโมเดลเราจะทำนาย test_data ได้ดี ต้องเอาไปทดสอบอีกที



Caret Interface Summary

Classification vs. Regression

Classification

```
set.seed(42)

ctrl <- trainControl(method = “cv”,
                     number = 5)

model <- train(
     y ~ .,
     data = df,
     method = “knn”,
     metric = “Accuracy”,
     trControl = ctrl
)
```

Regression

```
set.seed(42)

ctrl <- trainControl(method = “cv”,
                     number = 5)

model <- train(
     y ~ .,
     data = df,
     method = “knn”,
     metric = “RMSE”,
     trControl = ctrl
)
```

Same interface, different metrics

Classification Interfaces

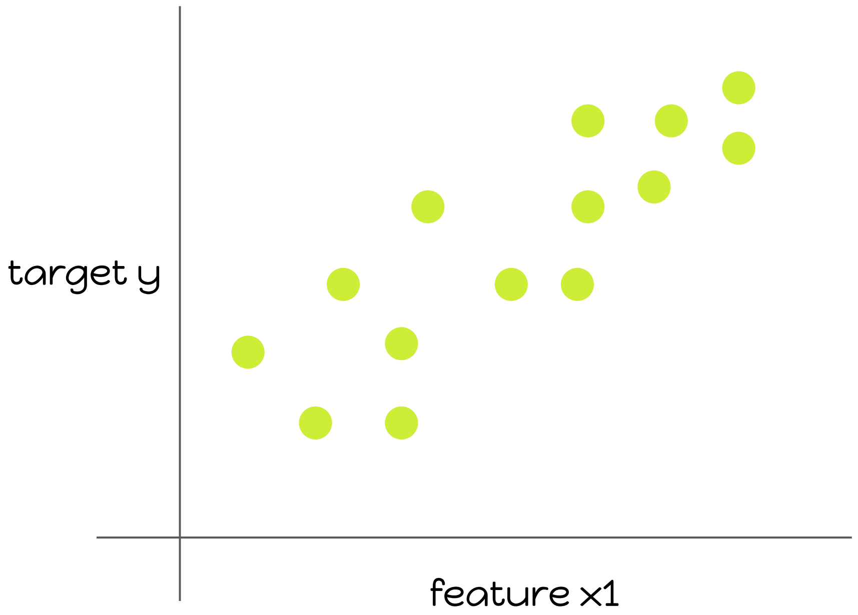
Classification - ROC Sens Specs

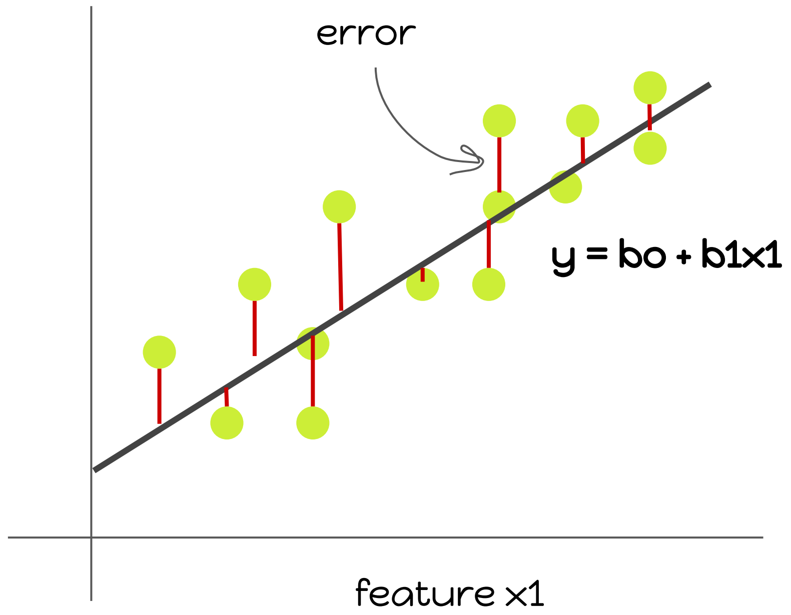
```
set.seed(42)

ctrl <- trainControl(
      method = “cv”,
      number = 5,
      summaryFunction = twoClassSummary,
      classProbs = TRUE)

model <- train(
      y ~ .,
      data = df,
      method = “knn”,
      metric = “ROC”,
      trControl = ctrl
)
```

Classification - AUC Precision Recall F1

```
set.seed(42)

ctrl <- trainControl(
      method = “cv”,
      number = 5,
      summaryFunction = prSummary,
      classProbs = TRUE)

model <- train(
      y ~ .,
      data = df,
      method = “knn”,
      metric = “AUC”,
      trControl = ctrl
)
```

Linear Regression Explained





Linear Regression Model

$$\hat{y} = B_0 + B_1 X_1$$

prediction

y intercept

slope

Linear Regression finds B0 and B1 that minimize the error

Minimize Error

$$minimize \sum (prediction - actual)^2$$

$$minimize \sum (\hat{y} - y)^2$$

Sum of Squared Error
or RSS (for short)

Common Regression Metrics

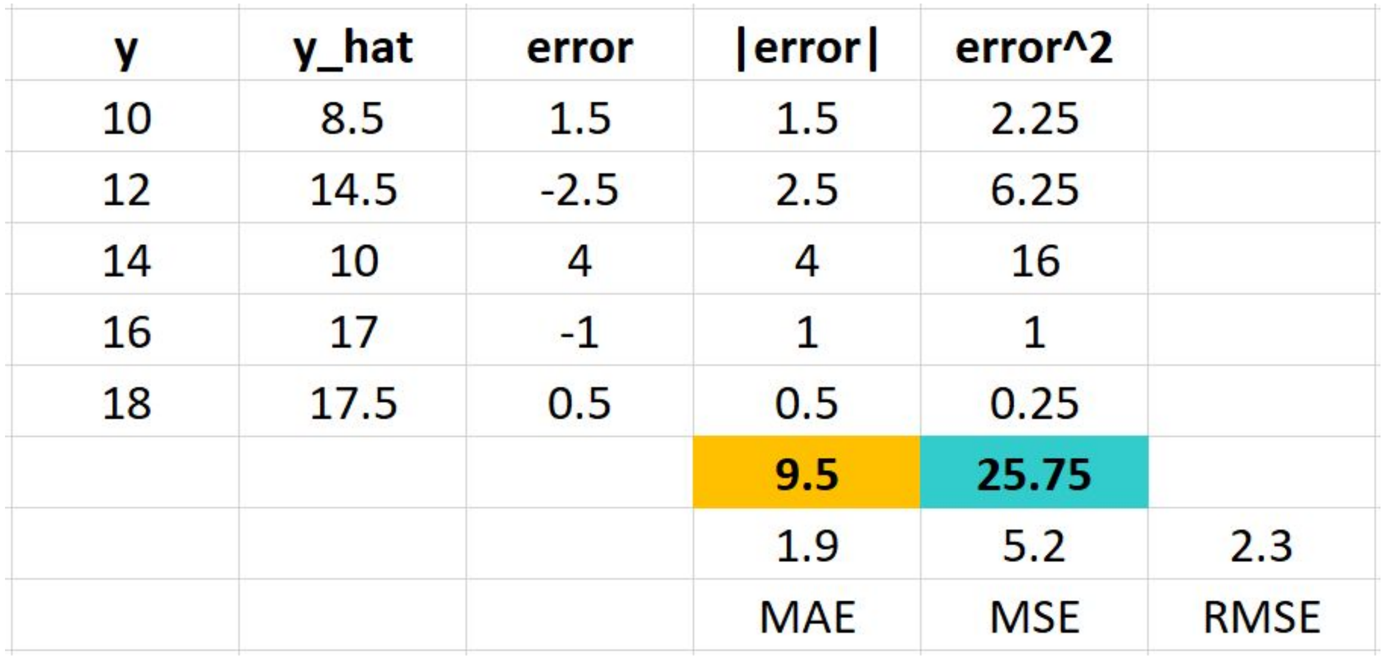
$$MAE = \frac{1}{n} * \sum |\hat{y} - y|$$

$$MSE = \frac{1}{n} * \sum (\hat{y} - y)^2$$

$$RMSE = \sqrt{\frac{1}{n} * \sum (\hat{y} - y)^2}$$

โมเดลที่เราเทรนจะพยายามทำให้ค่า MAE/ MSE/ RMSE มีค่าต่ำที่สุด i.e. minimize error

Easy to compute in Spreadsheets



y	y_hat	error	|error|	error^2	
10	8.5	1.5	1.5	2.25	
12	14.5	-2.5	2.5	6.25	
14	10	4	4	16	
16	17	-1	1	1	
18	17.5	0.5	0.5	0.25	
			9.5	**25.75**	
			1.9	5.2	2.3
			MAE	MSE	RMSE

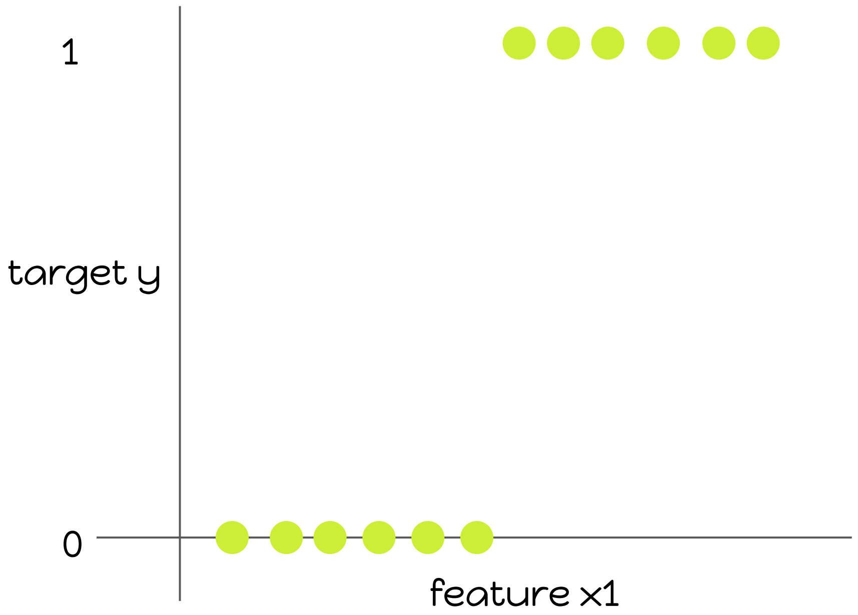
Build linear regression in R

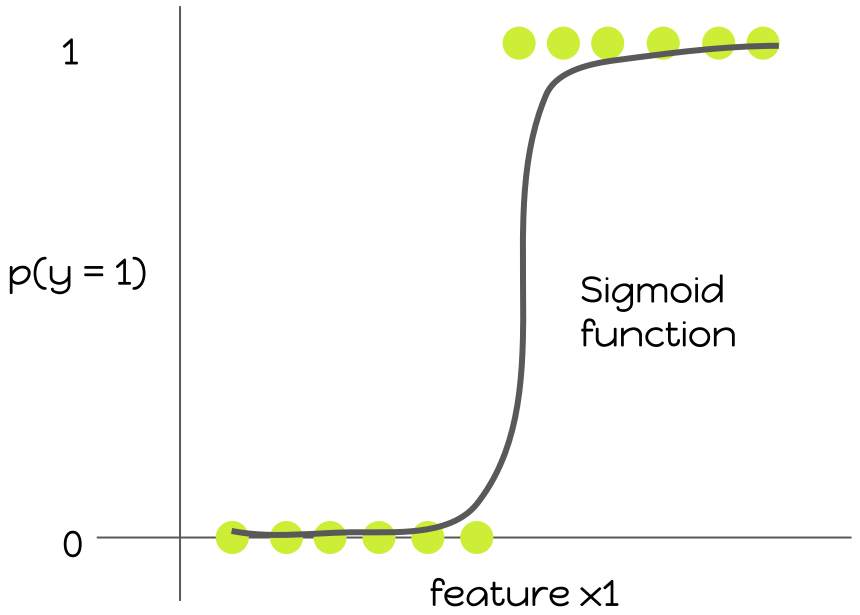
```
## train model with train_data
set.seed(99)
lm_model <- train(medv ~ rm + indus + crim,
                  data = train_data,
                  method = "lm")

## test model (predict test data)
p <- predict(lm_model, newdata = test_data)
rmse <- sqrt(mean( (p - test_data$medv)** 2 ))
```

Linear regression with K-Fold

```
## train model with train_data
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5,
                     verboseIter = TRUE)

lm_model <- train(medv ~ rm + indus + crim,
                  data = train_data,
                  method = "lm",
                  trControl = ctrl)

## test model (predict test data)
p <- predict(lm_model, newdata = test_data)
rmse <- sqrt(mean( (p - test_data$medv)** 2 ))
```

Logistic regression for binary classification





Logistic is very similar to linear regression

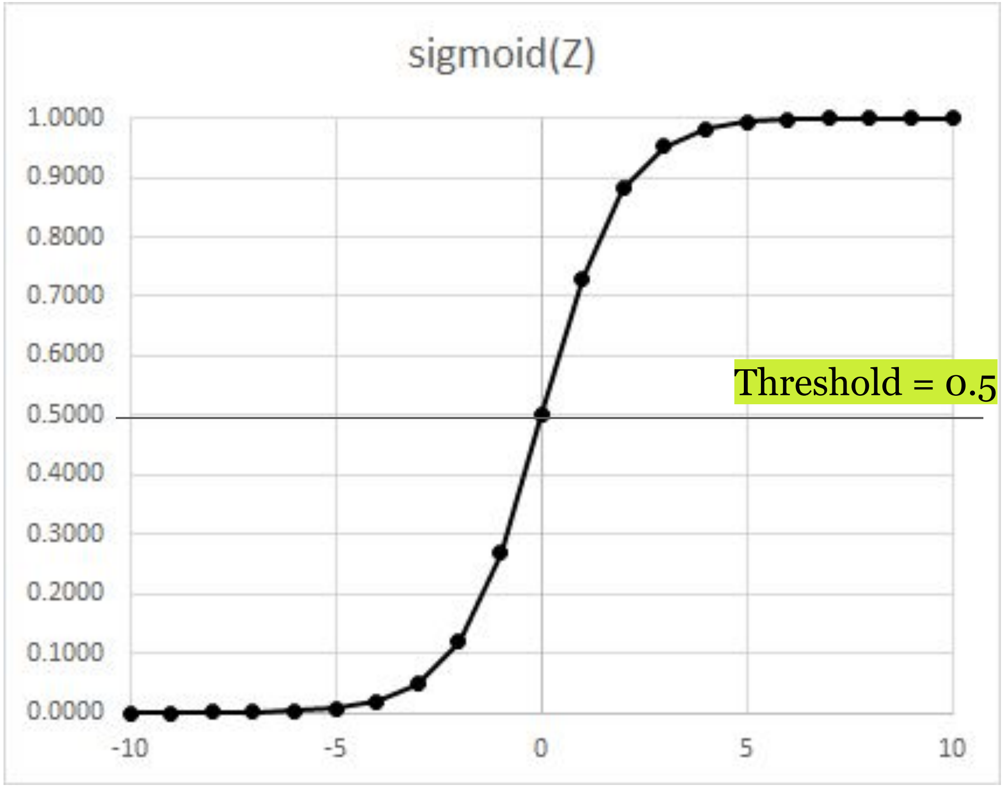
$$\boxed{Z} = B_0 + B_1X_1$$

$$P(Y = 1|x) = \frac{e^{Z}}{1 + e^{Z}}$$

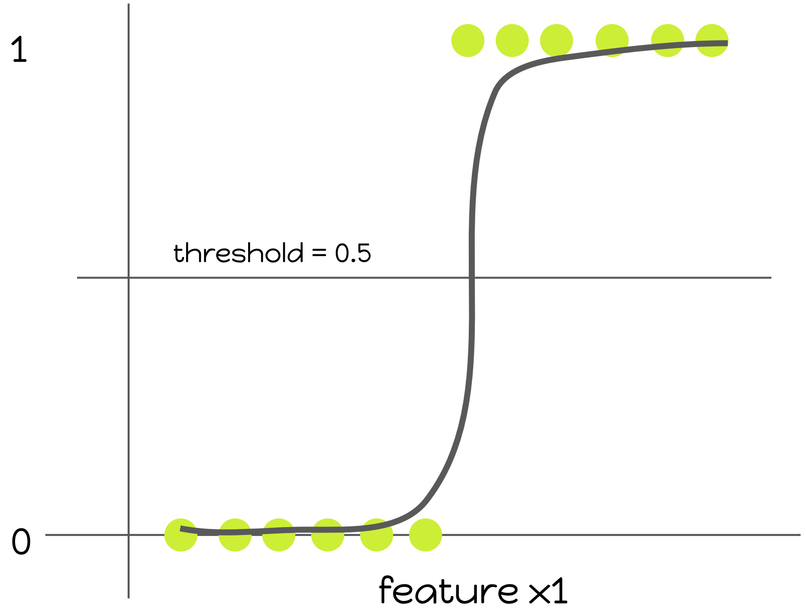
Sigmoid function

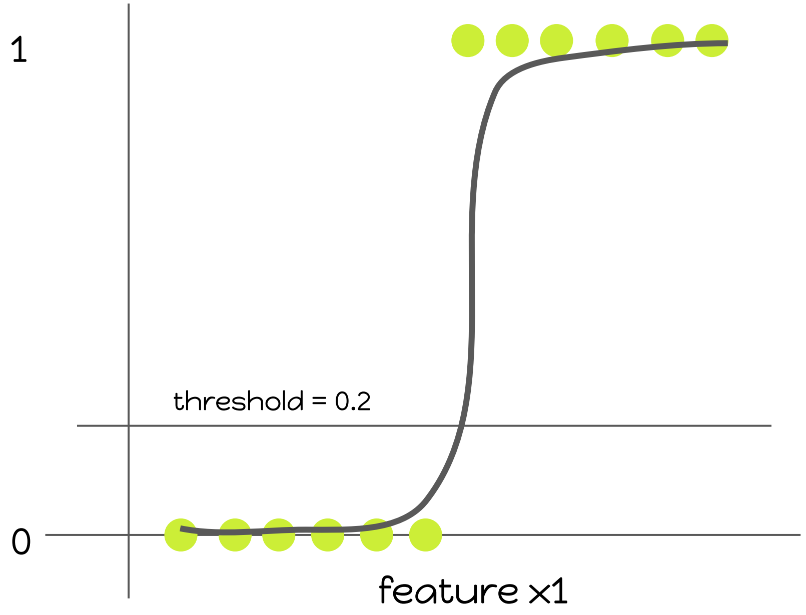
Z	sigmoid(Z)	y_hat
-10	0.0000	0
-9	0.0001	0
-8	0.0003	0
-7	0.0009	0
-6	0.0025	0
-5	0.0067	0
-4	0.0180	0
-3	0.0474	0
-2	0.1192	0
-1	0.2689	0
0	0.5000	0
1	0.7311	1
2	0.8808	1
3	0.9526	1
4	0.9820	1
5	0.9933	1
6	0.9975	1
7	0.9991	1
8	0.9997	1
9	0.9999	1
10	1.0000	1

If sigmoid(Z) > 0.5, predict y = 1, else y = 0



Our prediction changes if threshold changes





R Code

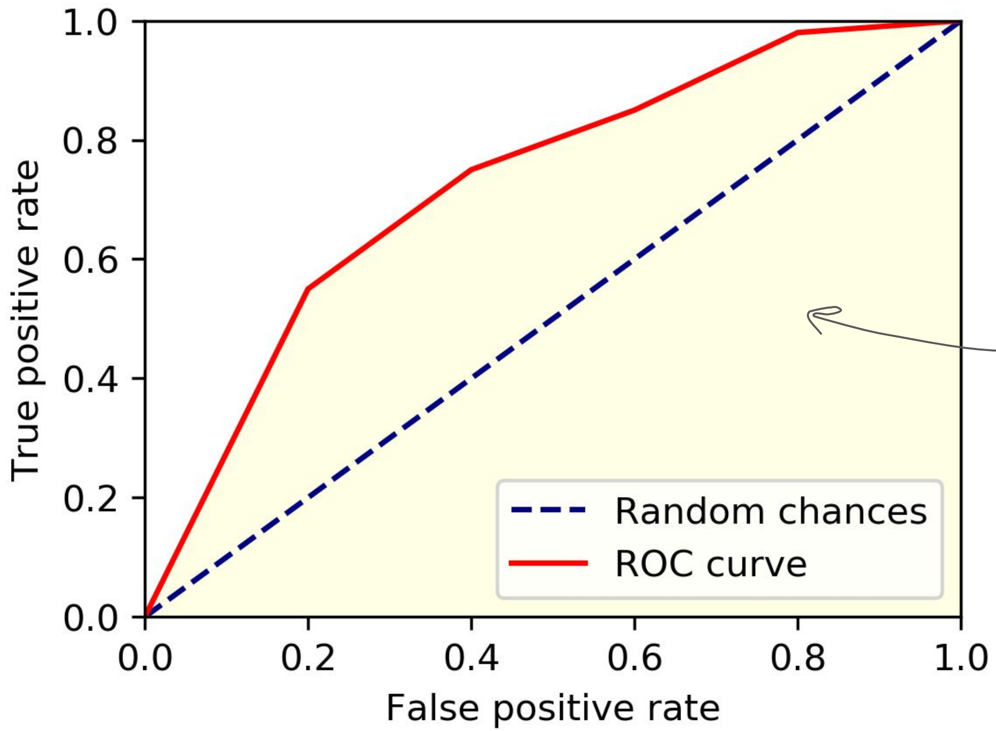
```
## train model with train_data
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5,
                     verboseIter = TRUE)

logistic_model <- train(diabetes ~ .,
                        data = train_data,
                        method = "glm",
                        trControl = ctrl)

## test model (predict test data)
p <- predict(logistic_model, newdata = test_data)
accuracy <- mean(p == test_data$diabetes)
```

Common classification metrics

- Accuracy
- Precision
- Recall
- F1

สามารถคำนวณได้
ง่ายๆจาก
Confusion Matrix

Common classification metrics



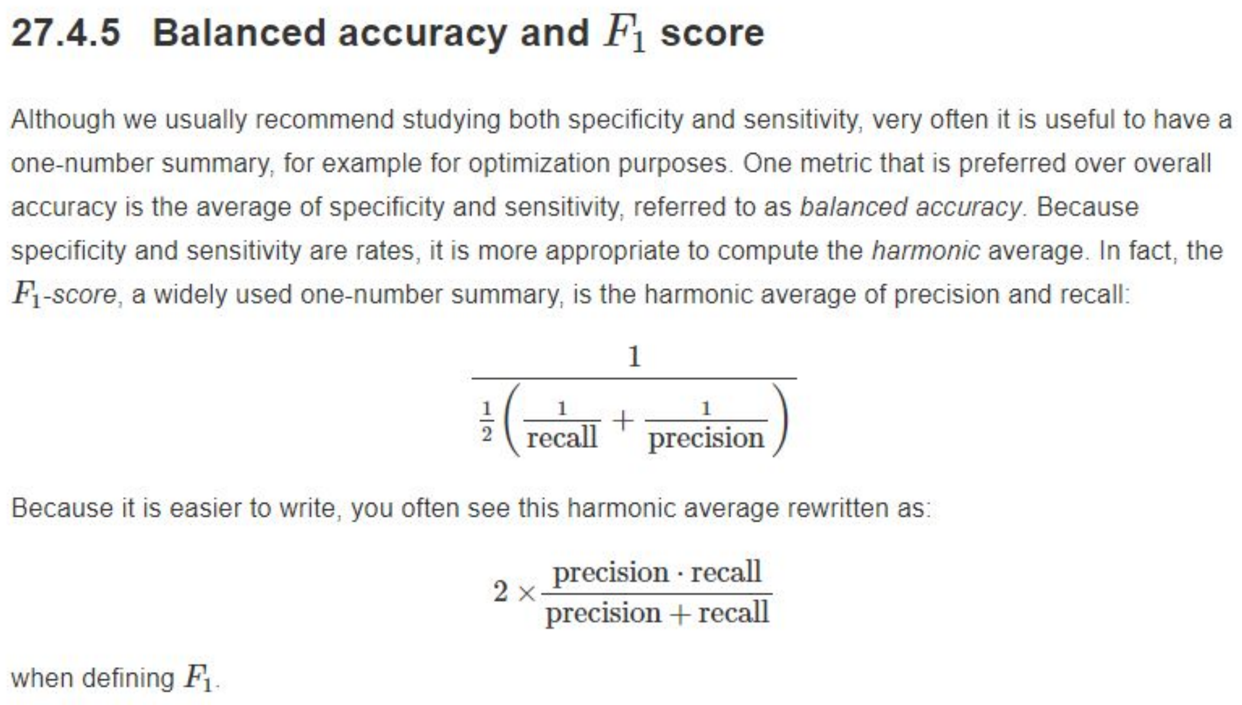
AUC

Area Under Curve

Interpretation in Thai

Metrics	**ความหมาย**
Accuracy	ความถูกต้องของโมเดลในภาพรวม
Precision	ทุก 100 ครั้งที่เราทำนาย y=1 โอกาสถูกเท่าไร
Recall	ทุกผู้ป่วยจริงๆ 100 คน เราตรวจเจอกี่คน
F1	ค่าเฉลี่ยระหว่าง precision, recall

F1 Score



27.4.5 Balanced accuracy and F_1 score

Although we usually recommend studying both specificity and sensitivity, very often it is useful to have a one-number summary, for example for optimization purposes. One metric that is preferred over overall accuracy is the average of specificity and sensitivity, referred to as *balanced accuracy*. Because specificity and sensitivity are rates, it is more appropriate to compute the *harmonic* average. In fact, the F_1-*score*, a widely used one-number summary, is the harmonic average of precision and recall:

$$\frac{1}{\frac{1}{2}\left(\frac{1}{\text{recall}} + \frac{1}{\text{precision}}\right)}$$

Because it is easier to write, you often see this harmonic average rewritten as:

$$2 \times \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

when defining F_1.

[Chapter 27 Introduction to machine learning | Introduction to Data Science (rafalab.github.io)]

R Code - Confusion Matrix

```
## use table()
table(predicted, actual, dnn = c(“predicted”, “actual”))
```

Actual

Predicted

	neg	pos
neg	101	33
pos	14	44

```
## how we calculate four metrics
accuracy <- (101 + 44) / (101 + 33 + 44 + 14)
precision <- 44 / (44 + 14)
recall <- 44 / (44 + 33)
F1 <- 2 * (precision * recall) / (precision + recall)
```

Regularized Regression

1. Ridge Regression
2. Lasso Regression

Regularization is a key technique in ML to reduce overfitting :D

Ridge Regression (L2)

$$RSS = \sum (\hat{y} - y)^2$$

Normal RSS from Linear Regression

$$Ridge\ RSS = \sum (\hat{y} - y)^2 + \boxed{\lambda \sum \beta^2}$$

Ridge add this term to the error function

Lasso Regression (L1)

$$RSS = \sum (\hat{y} - y)^2$$

Normal RSS from Linear Regression

$$Lasso\ RSS = \sum (\hat{y} - y)^2 + \boxed{\lambda \sum |\beta|}$$

Lasso add this term to the error function

Regularization helps reduce overfitting

```
y_hat  = bo + b1x1 + b2x2 + b3x3 + b4x4
y_hat  = 100 + 150x1 + 200x2 + 120x3 + 80x4
```

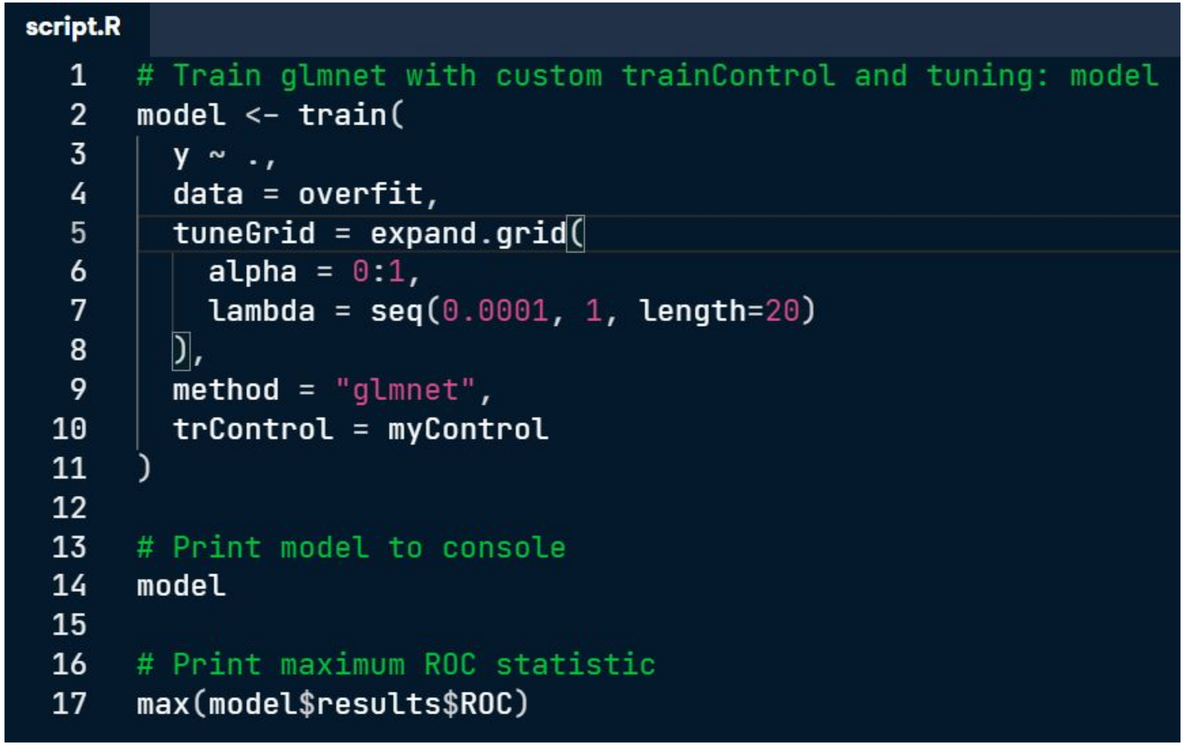
Lasso = RSS + Lambda * (150 + 200 + 120 + 80)

Lambda = 0
Error is the same as Linear Regression

Lambda > 0
All coefficient (B) in the model must be shrunken to reduce the new error

Ridge and Lasso in R

Machine Learning with caret in R | DataCamp


script.R

```
# Train glmnet with custom trainControl and tuning: model
model <- train(
  y ~ .,
  data = overfit,
  tuneGrid = expand.grid(
    alpha = 0:1,
    lambda = seq(0.0001, 1, length=20)
  ),
  method = "glmnet",
  trControl = myControl
)

# Print model to console
model

# Print maximum ROC statistic
max(model$results$ROC)
```

ElasticNet model

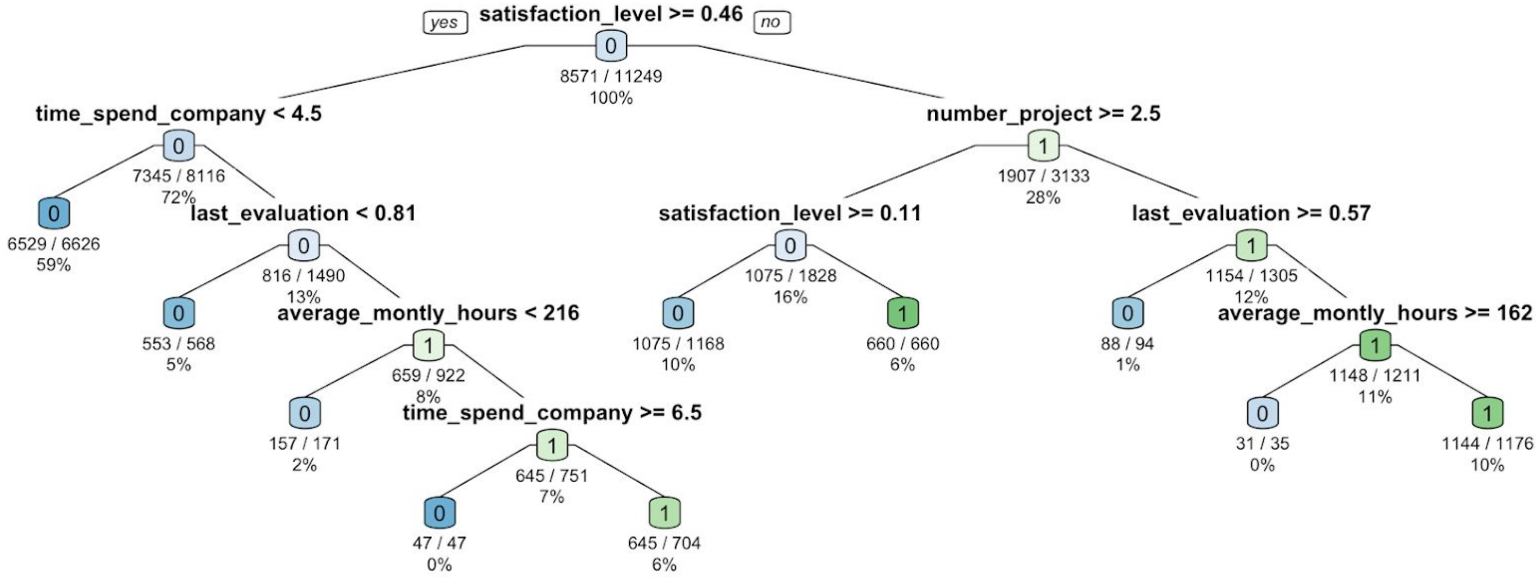
```
## train elasticnet model
set.seed(99)
ctrl <- trainControl(method = “cv”, number = 5,
                     verboseIter = TRUE)

enet_model <- train(diabetes ~ .,
                        data = train_data,
                        method = "glmnet",
                        trControl = ctrl)


## test model
p <- predict(enet_model, newdata = test_data)
accuracy <- mean(p == test_data$diabetes)
```

ElasticNet =
Mixed between
Ridge + Lasso

Time for a fun game :D

- Ask me a yes/no question
- To guess my favourite animal
- Max 10 questions

Decision Tree



How decision tree work?

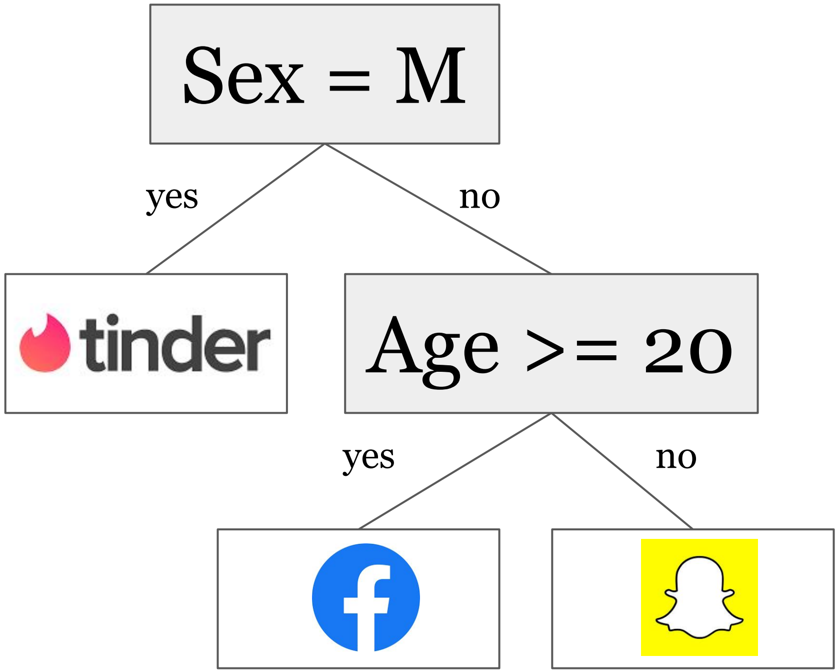
Age	Sex	App
15	M	tinder
20	M	tinder
16	F	👻
19	M	tinder
22	F	f
20	F	f

เวลาเราสร้าง decision tree เราถามคำถาม yes/no question ทีละข้อ

i.e. feature ใช้แบ่ง App ได้ดีที่สุด

How decision tree work?

Age	Sex	App
15	M	tinder
20	M	tinder
16	F	👻
19	M	tinder
22	F	f
20	F	f



Decision Tree with K-Fold

```
## train tree
set.seed(99)

ctrl <- trainControl(method = "....", number = ....,
                     verboseIter = TRUE)

tree_model <- train(diabetes ~ .,
                    data = ....,
                    method = "rpart",
                    trControl = ....)

## test model
p <- predict(tree_model, newdata = ....)
accuracy <- mean(....)
```

Let's do a quick review

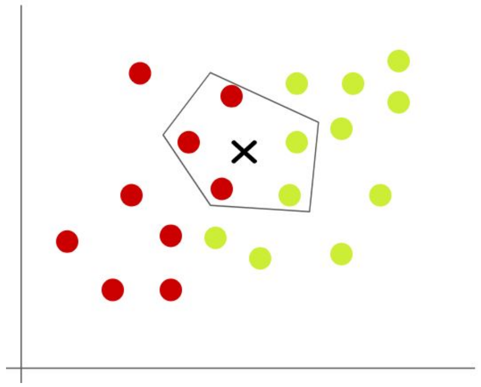
Parametric	**Non-Parametric**
Linear Regression	KNN
Logistic Regression	Decision Tree
Ridge Regression	Random Forest
Lasso Regression	

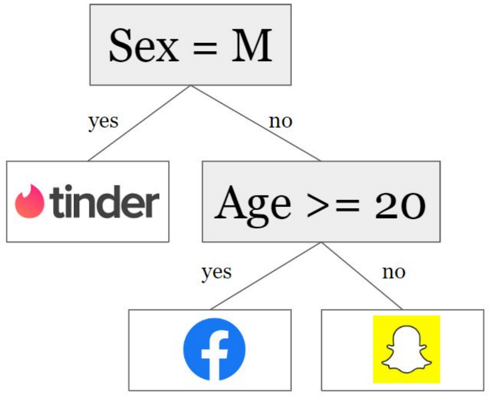
Regression is a Linear Combination (Parametric)

y_hat = bo + b1x1 + b2x2 + b3x3 + b4x4

มีฟอร์ม

While a non-parametric has no form :P





Random Forest

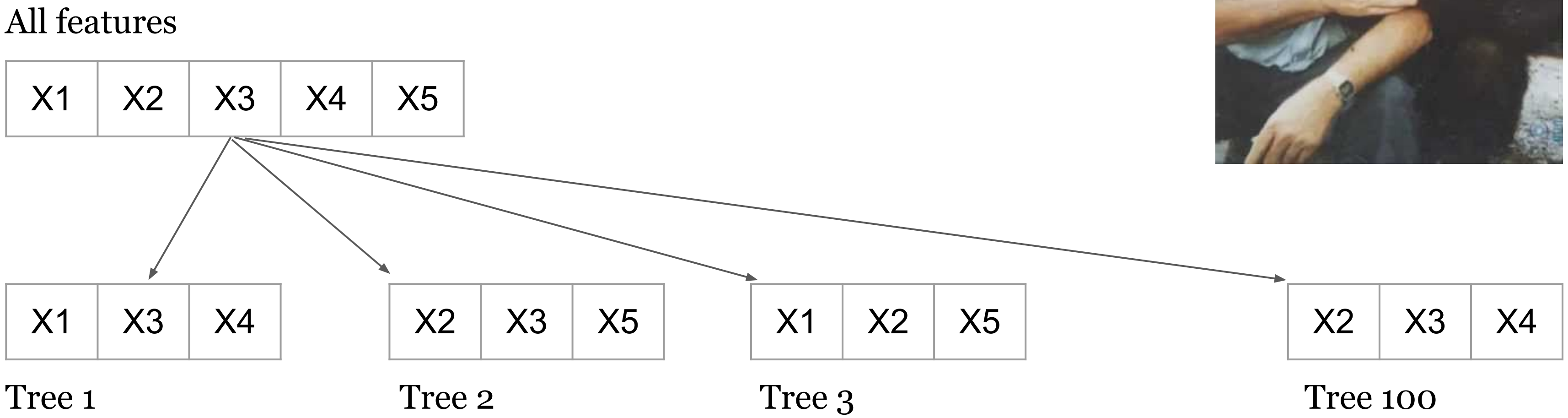
We grow hundreds of uncorrelated (decision) trees

$$\downarrow$$

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

Teamwork (Bagging)

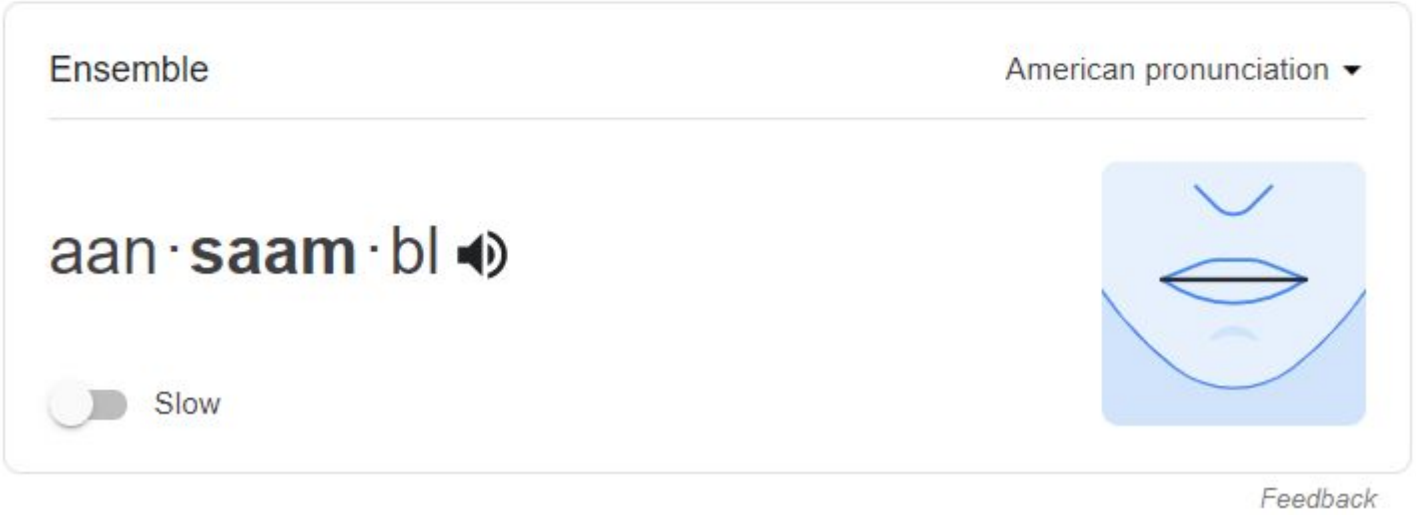
Bootstrap + mtry hyperparameter



Random Forest Code

```
## train random forests
set.seed(99)
myGrid <- expand.grid(mtry = 2:4)

ctrl <- trainControl(method = "....", number = ....,
                     verboseIter = TRUE)

rf_model <- train(diabetes ~ .,
                        data = ....,
                        method = "rf",
                         tuneGrid = myGrid,
                        trControl = ....)

## test model
p <- predict(rf_model, newdata = ....)
accuracy <- mean(....)
```

Ensemble Models


Ensemble

American pronunciation

aan·**saam**·bl

Slow

Feedback

อาน ซาม เบิ้ล

R **Ensemble Models**

นำโมเดลหลายๆตัวมาช่วยกันทำนายผล (Majority Vote)

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

Save our models for later use

```
saveRDS(model, “model.rds”)

model <- readRDS(“model.rds”)
```

Course Summary

- Machine Learning is **art + science**
- Try different models (No Free Lunch)
- Choose the simpler model
- Use CV + Grid Search to fine-tune model
- Start with Regression or decision tree because they are very fast to train

Bootcamp Live 07
Introduction to Machine Learning

Website: https://datarockie.com