Prediction

With Regression, Decision Trees, and Random Forests

Applied Machine Learning with R

www.therbootcamp.com

@therbootcamp

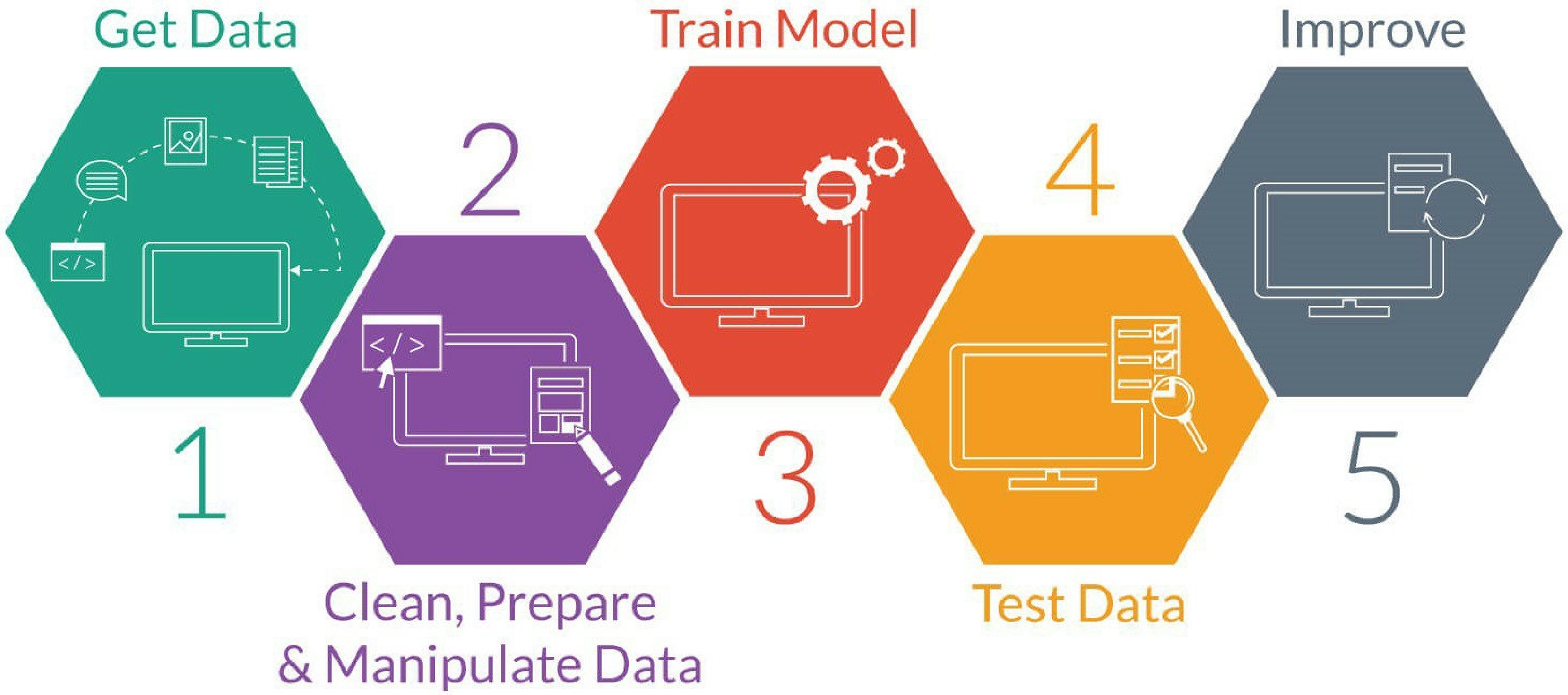
January 2019

Prediction is...

Nils Bohr, Nobel Laureate in Physics

Evan Esar

Anonymous



Source: Medium.com

What is model prediction?

Model prediction (aka, testing) is the process of computing a model's predictions on **test data**.

What is test data?

Test data is a separate, **'hold-out' data** set that the model **never saw during training**

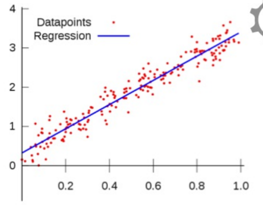
Model Training

Data Training

X1	X2	Y
129	0.23	390
154	0.12	459
153	0.24	461
174	0.13	523

Features Criterion

Model Training



$$Y = \beta_1 \cdot X1 + \beta_2 \cdot X2$$

$\beta_1 = 1$	$\beta_2 = 10$
$\beta_1 = 2$	**$\beta_2 = 20$**
$\beta_1 = 3$	$\beta_2 = 30$

Values Fitted

Y_Fit
391
464
464
525

Error Training

AE
1
5
3
2

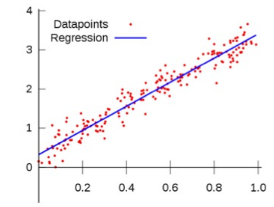
MAE Training
Mean Absolute Error

$$\frac{1+5+3+2}{4}$$

$$= 2.75$$

Model Testing

Model
Fixed!



$$Y = \beta_1 \cdot X1 + \beta_2 \cdot X2$$

$\beta_1 = 3$ **$\beta_2 = 20$**

$$Y = 3 \cdot X1 + 20 \cdot X2$$

Data
Testing

X1	X2	Y
143	0.19	420
139	0.08	412
124	0.32	382
148	0.39	435

Features Criterion

Values
Predicted

Y_Pred
433
419
378
455

Error
Testing

AE
13
7
4
20

MAE Testing
Mean Absolute Error

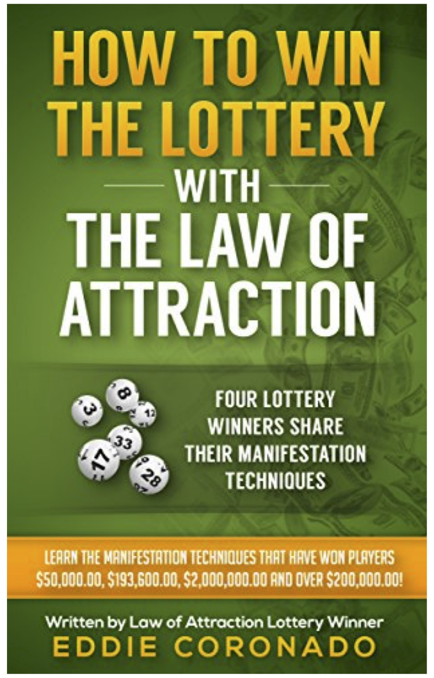
$$\frac{13 + 7 + 4 + 20}{4}$$

$$= 11$$

Why do we separate training from testing?

Just because a model can **fit past data well** (high training accuracy), does necessarily mean that it will **predict new data well** (high testing accuracy).

Evan Esar



Training data

id	sex	age	fam_history	smoking	criterion
1	m	45	No	FALSE	0
2	m	43	Yes	FALSE	1
3	f	40	Yes	FALSE	1
4	m	51	Yes	FALSE	1
5	m	44	No	TRUE	0

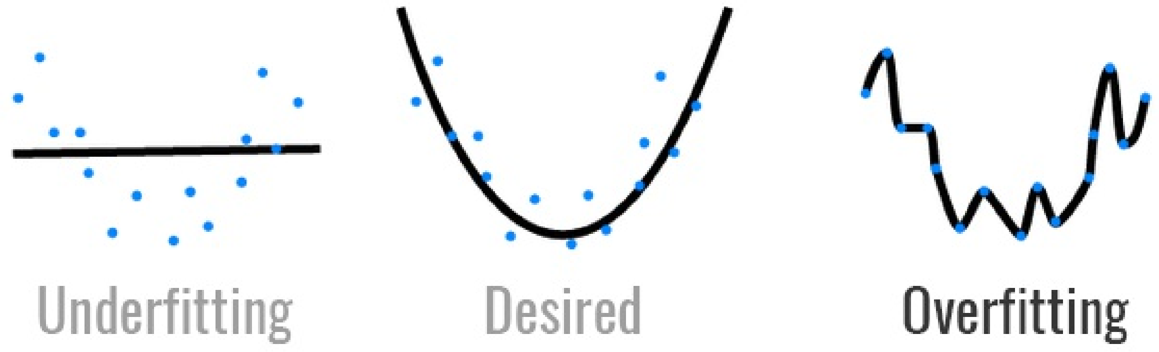
Test data

id	sex	age	fam_history	smoking	criterion
91	m	51	Yes	TRUE	?
92	f	47	No	TRUE	?
93	m	39	No	TRUE	?
94	f	51	Yes	TRUE	?
95	f	50	Yes	FALSE	?

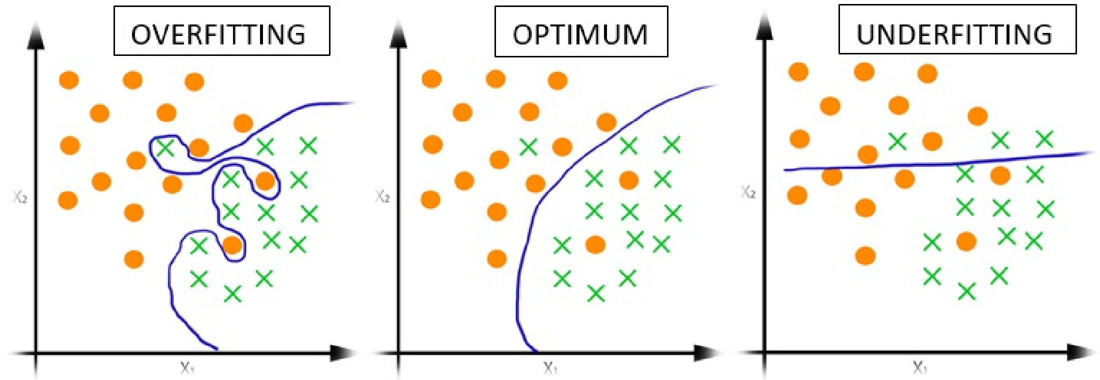
Overfitting

When a model is consistently **less accurate in predicting future data** than in **fitting training data**, this is called **overfitting**

Overfitting typically occurs when a model 'mistakes' random noise for a predictable signal

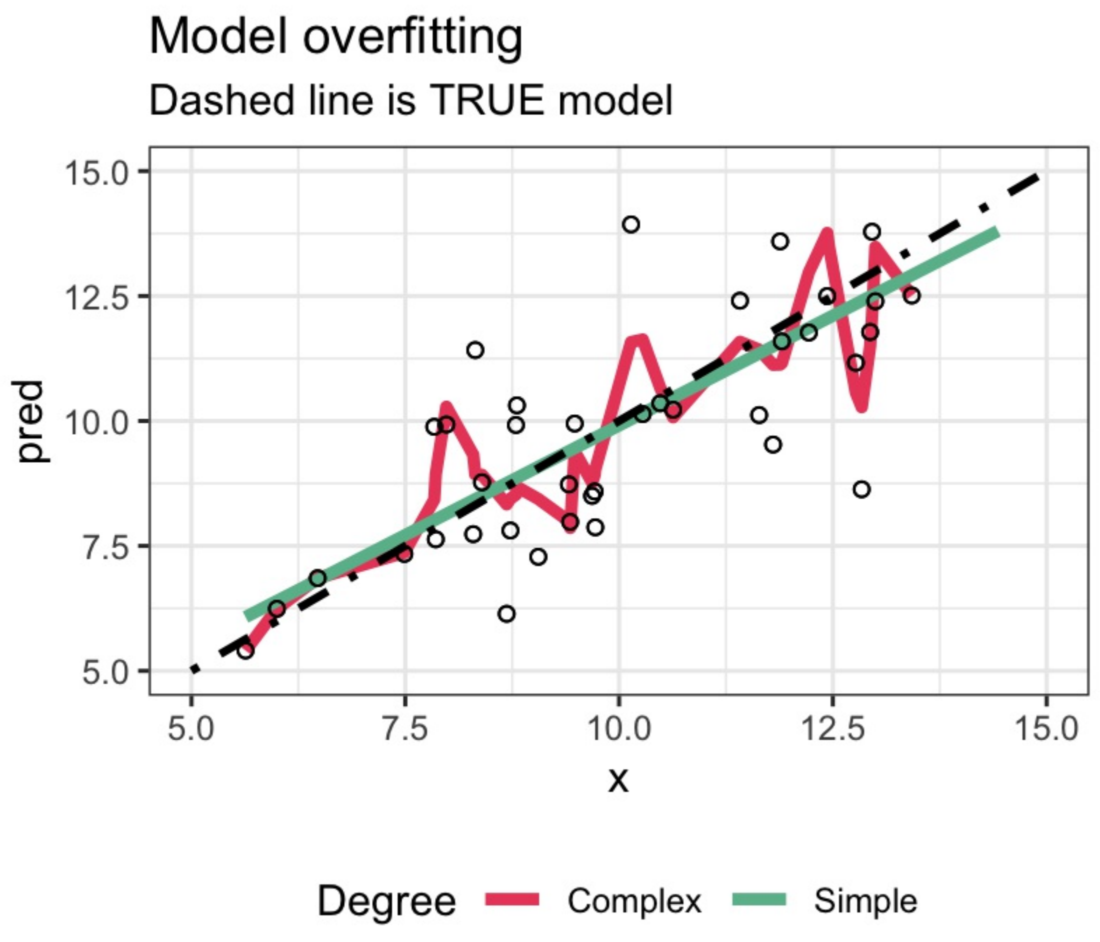


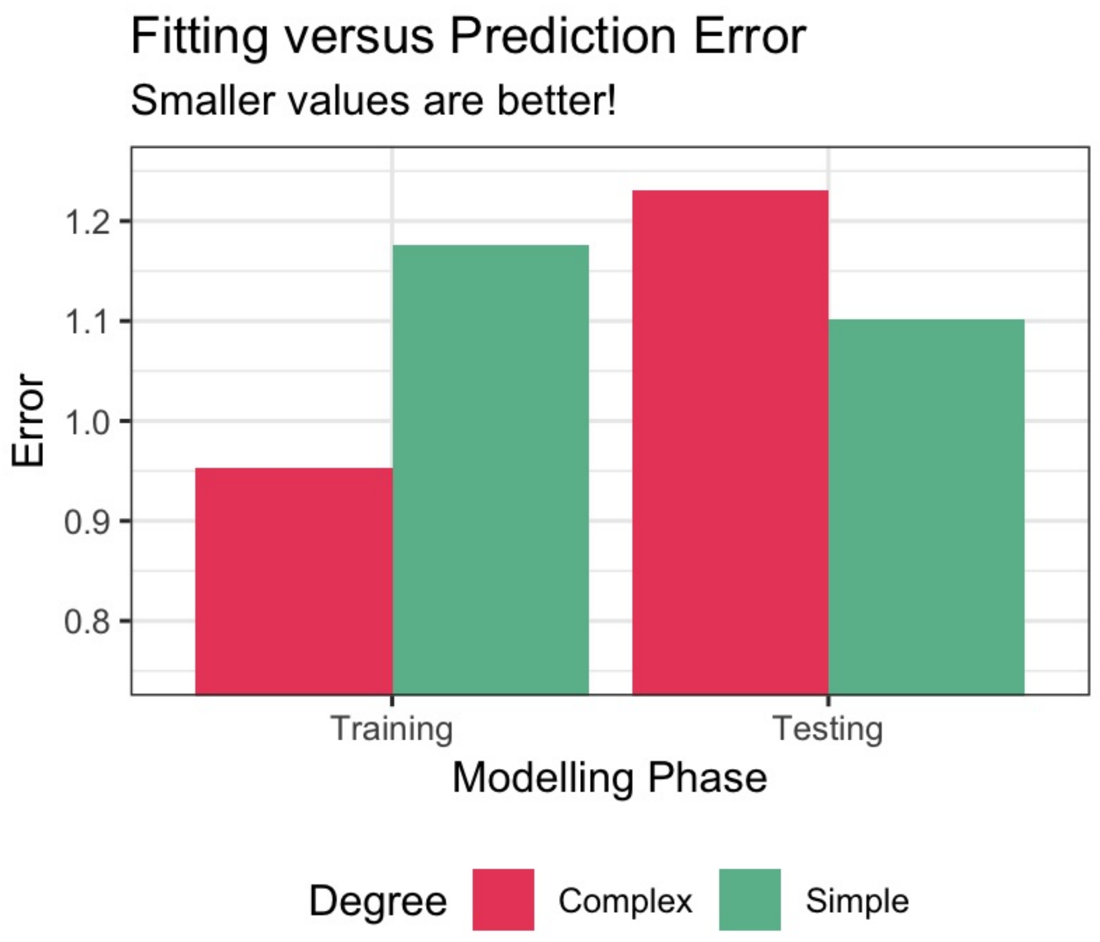
hackernoon.com



Medium.com

Overfitting





Overfitting

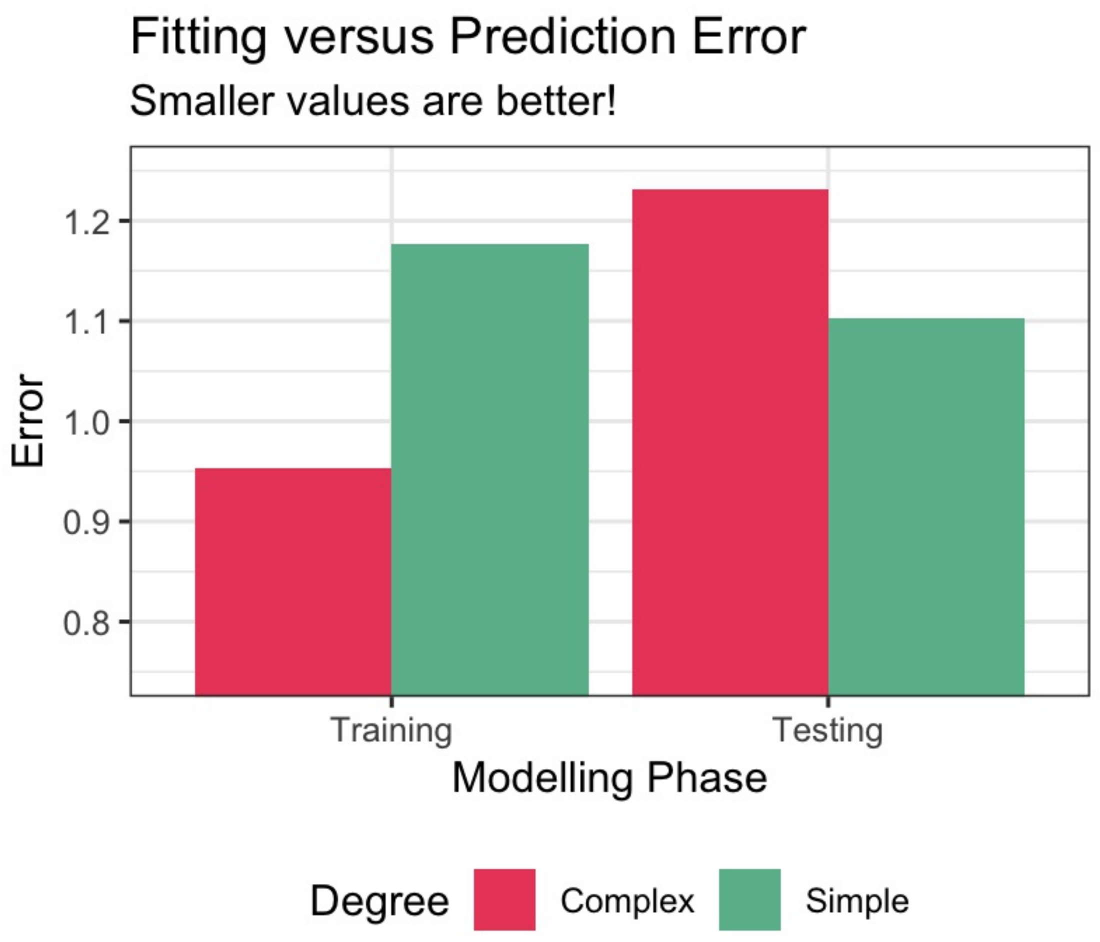
How do we account for overfitting?

Always evaluate models based on their performance on new, unseen test data

Use models with **regularization** terms, which explicitly punish models for being too complex.

Use fitting methods such as **cross-validation** to find optimal regularization values.

We will learn about these methods in a future session!



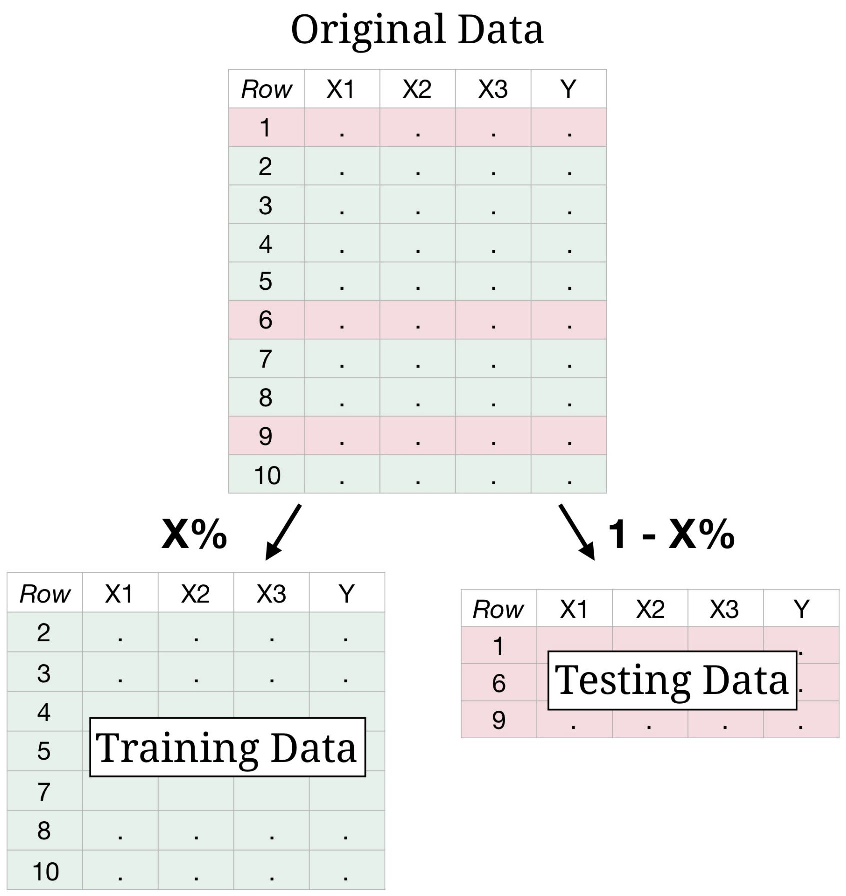
How do I get separate training and test data?

If you don't have two naturally occurring distinct training and test dataset, you can **randomly split** a dataset into an **X% training** set and **1-X% testing** set.

The `caret` function `createDataPartition()` helps you do this automatically.

Natural examples

Domain	Training	Test
Stock prediction	2017 Trends	2019 Trends
Medical diagnosis	Patients from Hospital A	Patients from Hospital B
Crime rates	Statistics from City X	Statistics from City Y



Two new models enter the ring...

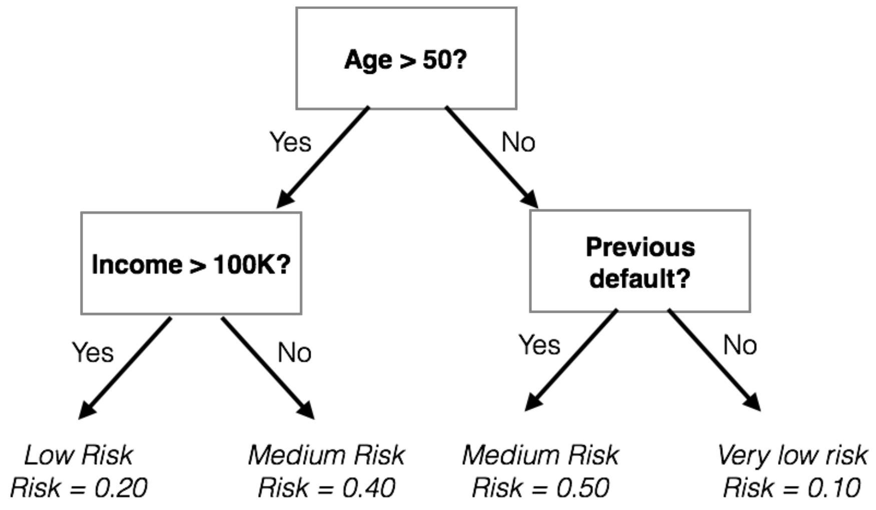
Regression

Decision Trees

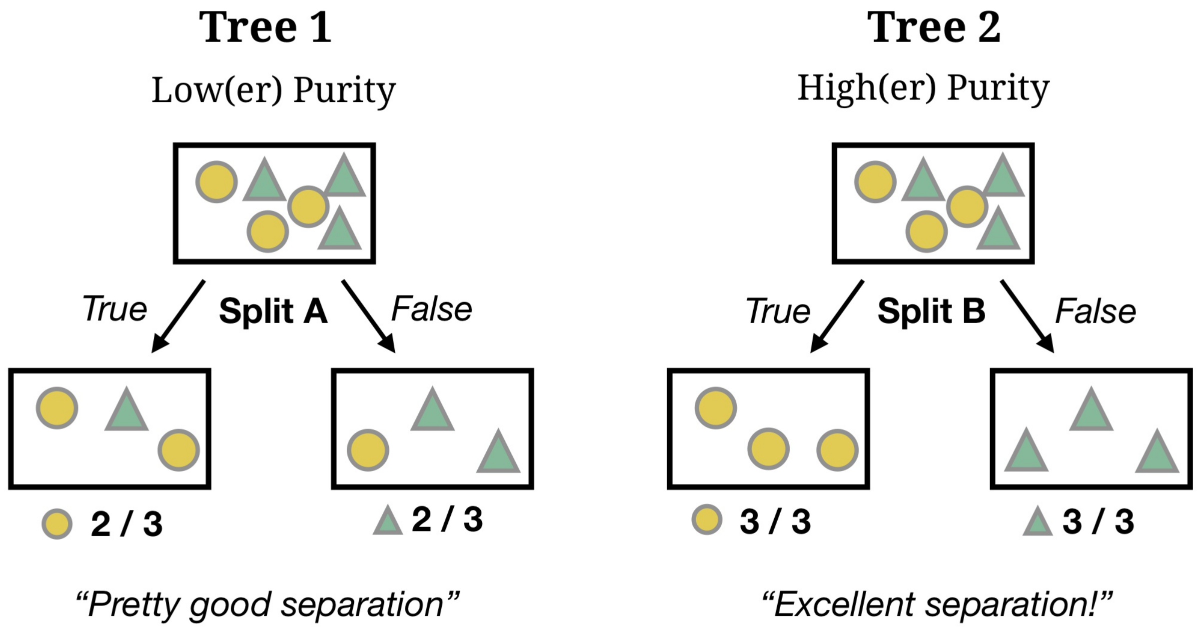
Random Forests

Decision Trees

In **decision trees**, the criterion is modeled as a **sequence of logical YES or NO questions**.

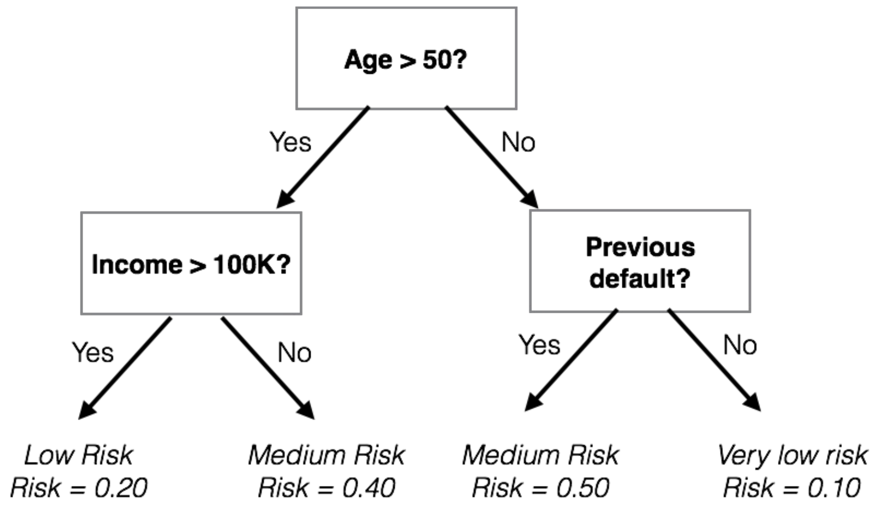


Grow Decisions Trees by splitting features that maximize
.



Decision Trees

In **decision trees**, the criterion is modeled as a **sequence of logical YES or NO questions**.



Fit a Decision Tree in `caret` using `method = "rpart"`.

```
# Fit a decision tree with a defined cp = .10

train(form = income ~ .,
      data = baselers,
      method = "rpart",  # Decision Tree
      trControl = ctrl,
      tuneGrid = expand.grid(cp = .10)) # cp
```

Decision Trees

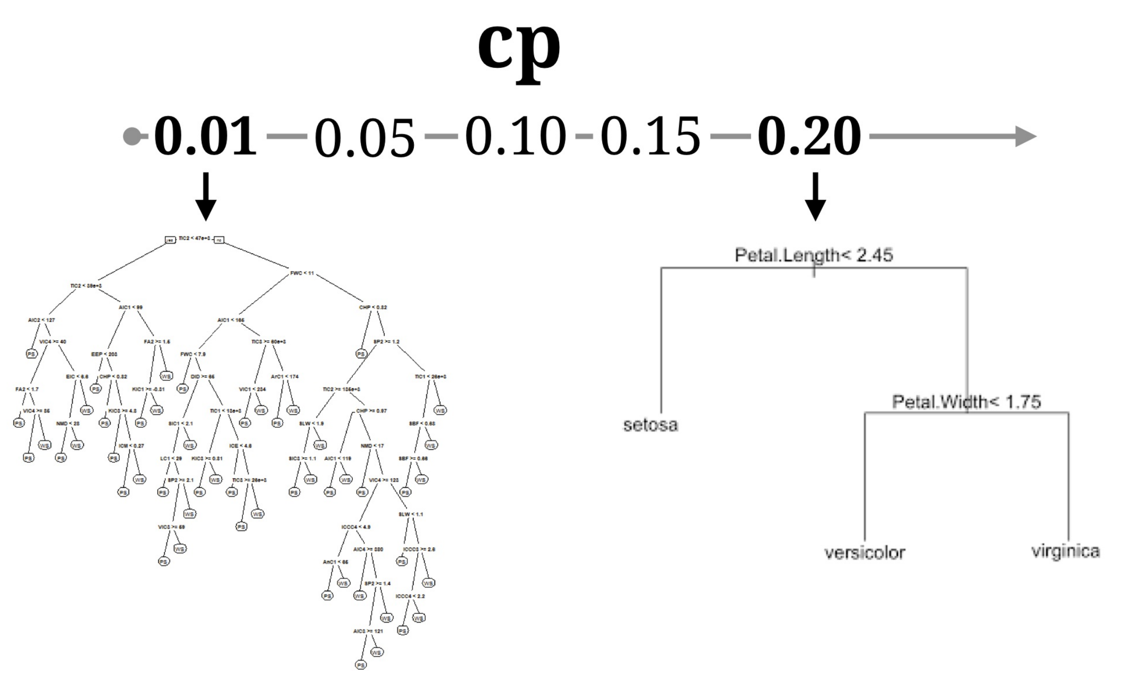
Complexity Parameter

Decision trees have a **complexity parameter** called `cp`.

The cp parameter controls how complex (i.e.; large) trees are allowed to grow

- **Small** cp (< 0.01) = **Complex** Trees
- **Large** cp (> 0.10) = **Simple** Trees

There is no "one" best value of cp -- the best value of cp depends on your needs and your dataset!



Small cp =
Complex tree

Large cp =
Simple tree

Decision Trees

Complexity Parameter

Decision trees have a **complexity parameter** called `cp`.

The `cp` parameter controls how complex (i.e.; large) trees are allowed to grow

- **Small** cp (< 0.01) = **Complex** Trees
- **Large** cp (> 0.10) = **Simple** Trees

There is no "one" best value of `cp` -- the best value of cp depends on your needs and your dataset!

Decision Trees in Caret: rpart

When fitting a decision tree, the `cp` parameter can be defined by the user in the `tuneGrid` argument:

```
# Fit a decision tree with a defined cp = .10

train(form = income ~ .,
      data = baselers,
      method = "rpart",  # Decision Tree
      trControl = ctrl,
      tuneGrid = expand.grid(cp = .10)) # cp
```

- `cp` can also be optimally determined through methods such as **cross-validation**, which we will learn later

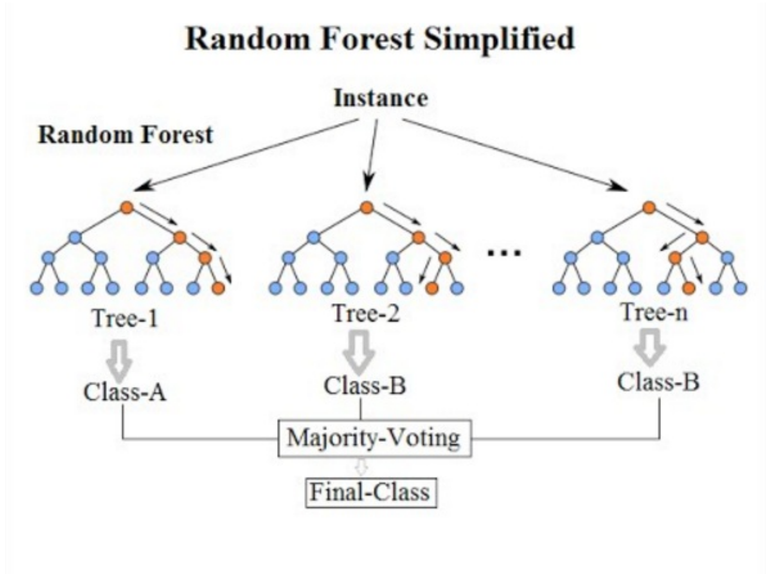
Regression

Decision Trees

Random Forests

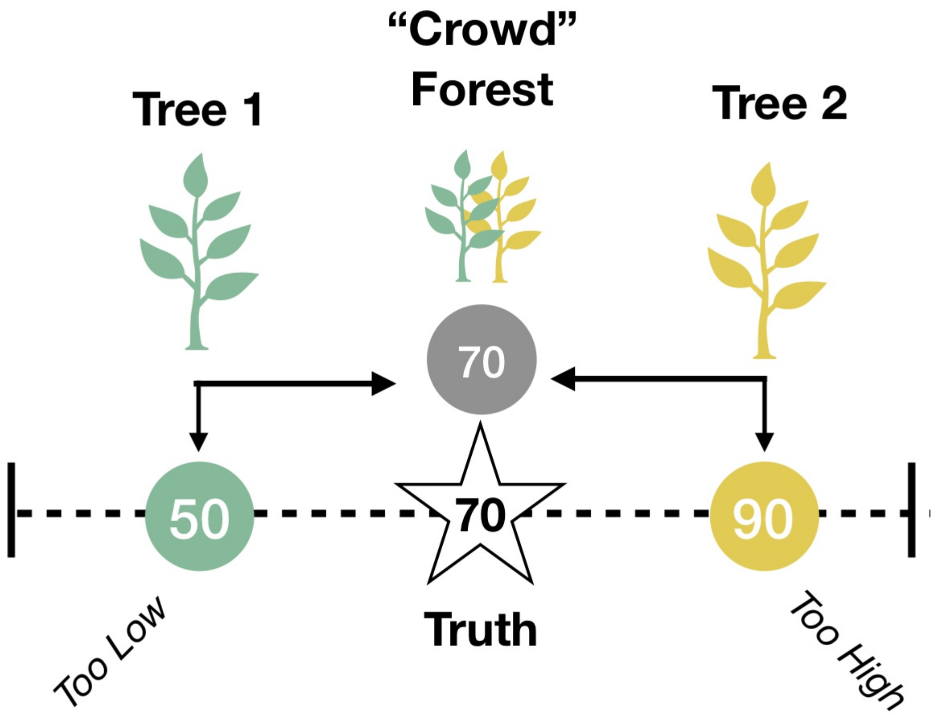
Random Forest

In Random Forest, the criterion is modeled as the **aggregate prediction of a large number of decision trees** each based on different features.



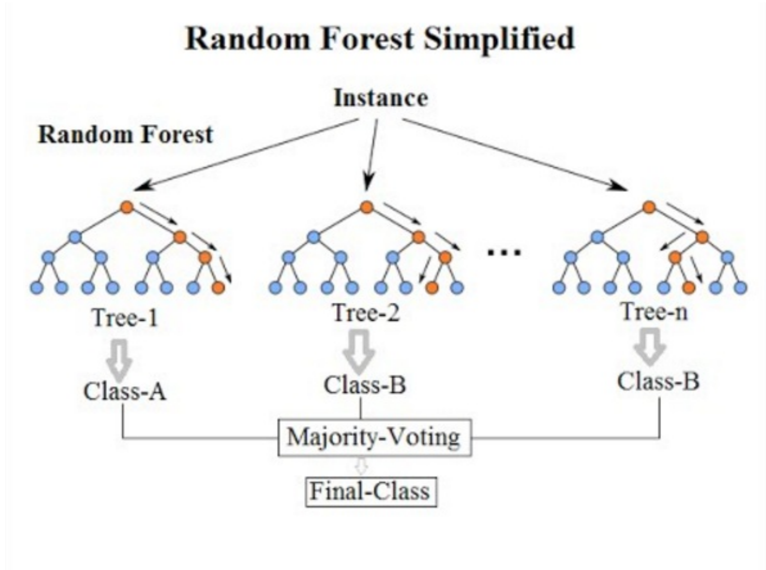
Source

In Random Forests, we create a large set of **diverse trees** that can be aggregated into one **Wisdom of Crowds** judgment.



Random Forest

In Random Forest, the criterion is modeled as the **aggregate prediction of a large number of decision trees** each based on different features.



Source

To **fit a random forest** in caret, use `method = "rf"`.

```
# Fit a random forest with a defined mtry = 3

train(form = income ~ .,
      data = baselers,
      method = "rf",  # Random Forest
      trControl = ctrl,
      tuneGrid = expand.grid(mtry = 3))
```

Random Forest

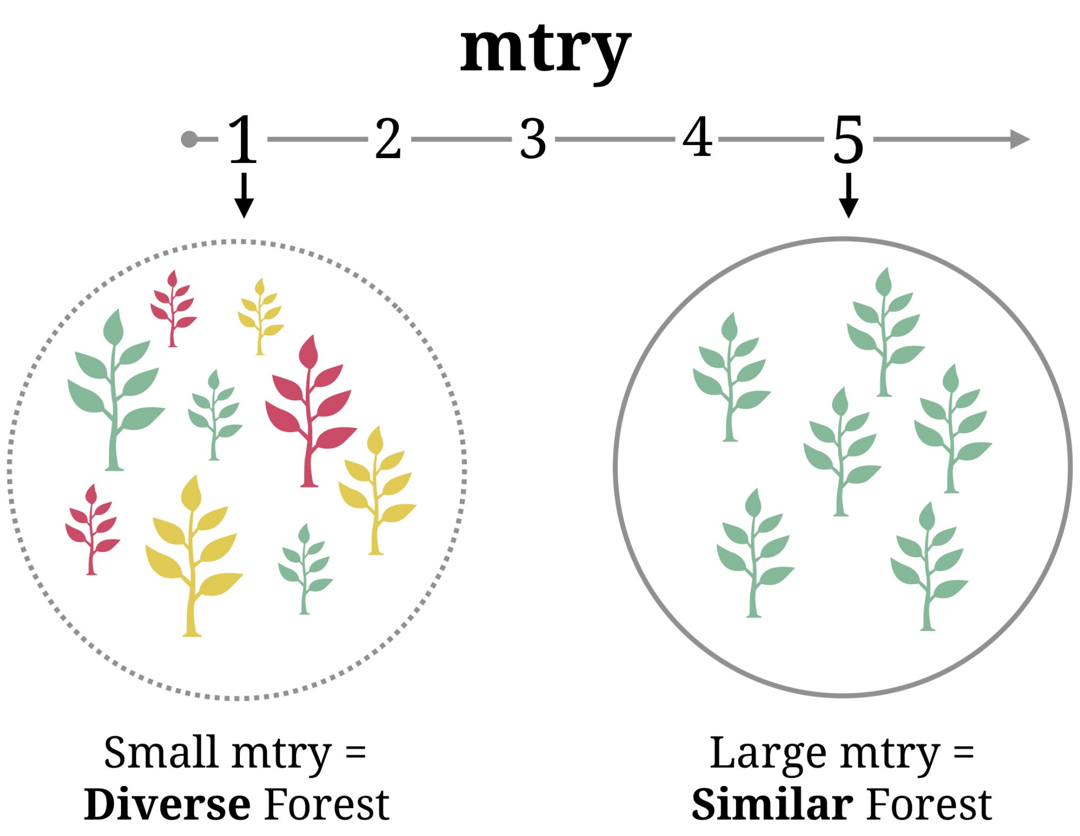
Diversity Parameter: mtry

Random Forests have a **diversity parameter** called **mtry**

Technically, this controls how many features are randomly considered at each split of the trees

- Small mtry (~ 1) = Diverse Forest
- Large mtry (> 5) = Similar Forest

There is no "one" best value of `mtry` -- the best value of `mtry` depends on your needs and your dataset!



Random Forest

Diversity Parameter: mtry

Random Forests have a **diversity parameter** called **mtry**

Technically, this controls how many features are randomly considered at each split of the trees

- Small mtry (~ 1) = Diverse Forest
- Large mtry (> 5) = Similar Forest

There is no "one" best value of `mtry` -- the best value of `mtry` depends on your needs and your dataset!

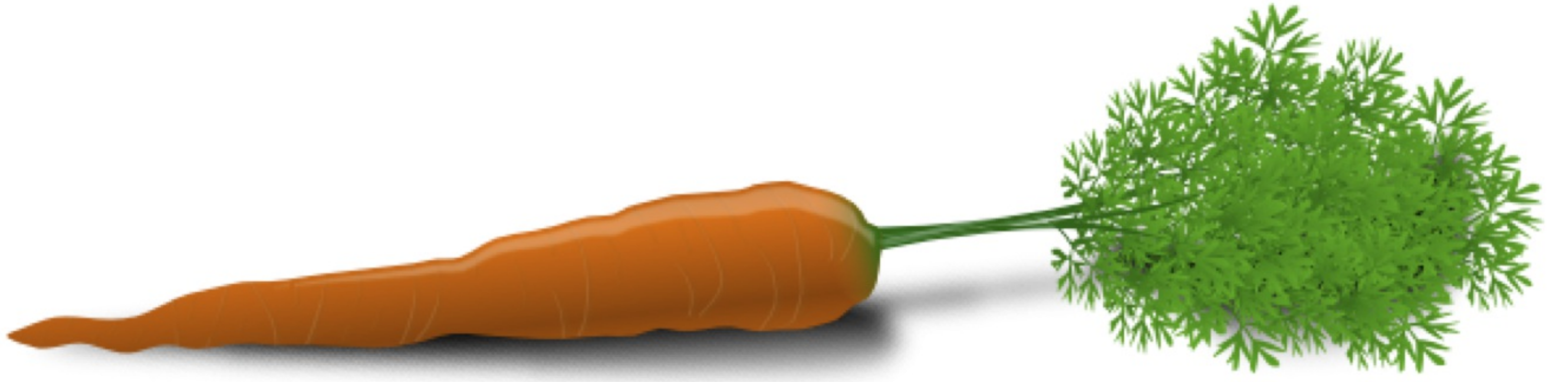
When fitting a random forest, the `mtry` parameter can be defined by the user in the `tuneGrid` argument.

```
# Fit a random forest with a defined mtry = 2

train(form = income ~ .,
      data = baselers,
      method = "rpart",   # Decision Tree
      trControl = ctrl,
      tuneGrid = expand.grid(mtry = 2)) # mtry
```

- `mtry` can also be optimally determined through methods such as **cross-validation**, which we will learn later

Evaluating model predictions with caret



Predict new data with predict()

To **test model predictions** with `caret`, all you need to do is get a vector of predictions from a new dataframe `newdata` using the `predict()` function:

```
# Get predictions for test data!
predict(mod, newdata = data_test)
```

argument	description
object	A machine learning / statistical object created from `caret`, ...
newdata	A dataframe of new data

This returns a vector of predicted values for your new data!

Get predictions, use `predict(mod, newdata = data_test)`

```
# Load training and test data
data_train <- read_csv("1_Data/XXX_train.csv")
data_test <- read_csv("1_Data/XXX_test.csv")

# Fit model to training data
mod <- train(form = Y ~ .,
             method = "glm",
             data = data_train)

# Get fitted values (for training data)
mod_fit <- predict(mod)

# Predictions for NEW data_test data!
mod_pred <- predict(mod, newdata = data_test)
```

Predict new data with predict()

To **test model predictions** with `caret`, all you need to do is get a vector of predictions from a new dataframe `newdata` using the `predict()` function:

```
# Get predictions for test data!
predict(mod, newdata = data_test)
```

argument	description
object	A machine learning / statistical object created from `caret`, ...
newdata	A dataframe of new data

This returns a vector of predicted values for your new data!

Compare predictions to the criterion with `postResample()`

```
# Define criterion
criterion_train <- data_train$Y
criterion_test <- data_test$Y

# Fitting performance
postResample(pred = mod_fit,
             obs = criterion_train)

#    RMSE Rsquared      MAE
#2.454015 0.848482 1.889584

# Prediction performance
postResample(pred = mod_pred,
             obs = criterion_test)

#     RMSE  Rsquared       MAE
#3.4763941 0.6977009 2.6764346
```

Split data with createDataPartition()

Use `createDataPartition()` to **split a dataset** into separate training and test datasets

```
# Create a set of indices for random
# selection of 70% of data

createDataPartition(y = data$Y
                    p = .7,
                    list = FALSE)
```

Argument	Description
y	The criterion
p	Percent of data to select

This returns a vector of indices you can then use to select rows (see right)

Create separate `XX_train` and `data_test` datasets from a single 'large' dataset

```
# Set the randomisation seed to get the
#  same results each time
set.seed(100)

# Get indices for training
index <- createDataPartition(y = baselers$income,
                             p = .7,
                             list = FALSE)

# Create training data
baselers_train <- baselers %>%
  slice(index)

# Create test data
baselers_test <- baselers %>%
  slice(-index)
```

5 steps with caret

Step 0: Load training and test data (or create with `createDataPartition()`)

```
data_train <- read_csv("1_Data/XXX_train.csv")
data_test <- read_csv("1_Data/XXX_test.csv")
```

Step 1: Define control parameters

```
# Use method = "none" for no advanced fitting
ctrl <- trainControl(method = "none")
```

Step 2: Train model

```
mod <- train(form = Y ~ .,
             data = data_train,
             method = "My Favorite Model",
             trControl = ctrl,
             tuneGrid = expand.grid(mtry = 2))
```

Step 3: Explore

```
mod            # Print object
mod$finalModel # Final model
```

Step 4: Predict

```
rpart_pred <- predict(object = mod,
                      newdata = data_test)
```

Step 5: Evaluate prediction accuracy

```
postResample(pred = rpart_pred,
             obs = data_test$Y)
```

Questions?

Practical