Model Fitting

With Regression

Applied Machine Learning with R

www.therbootcamp.com

@therbootcamp

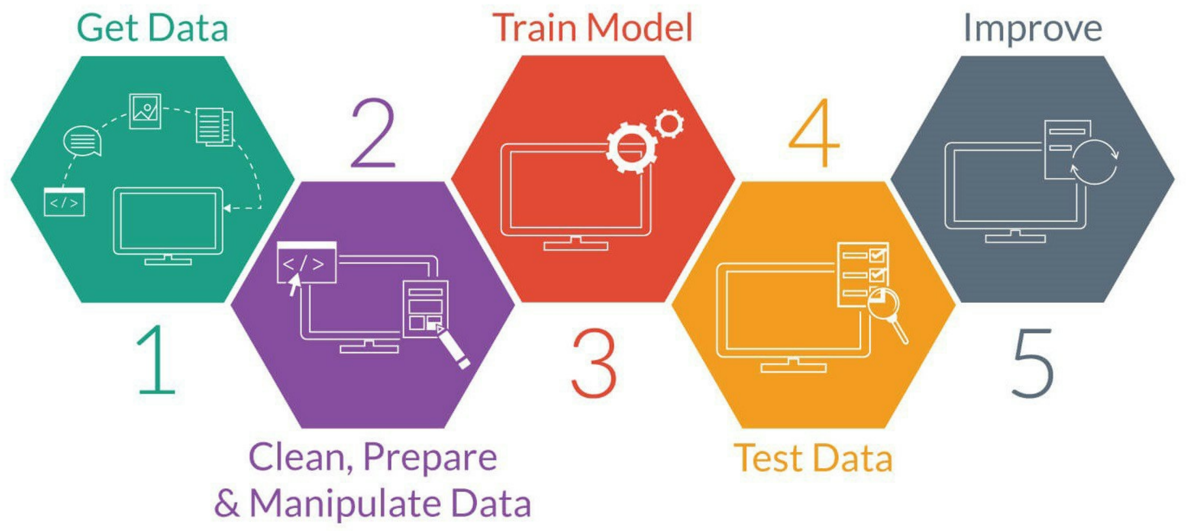
January 2019

Where we are at

- Have a business **question**: How can I predict loan default?
- Have **data** relevant to that question: Records from 300 historical customers.
- Data is cleaned and in a **tidy**, rectangular format: Database, .csv.

What's next?

Select and **train** model(s) depending on the **type of task**



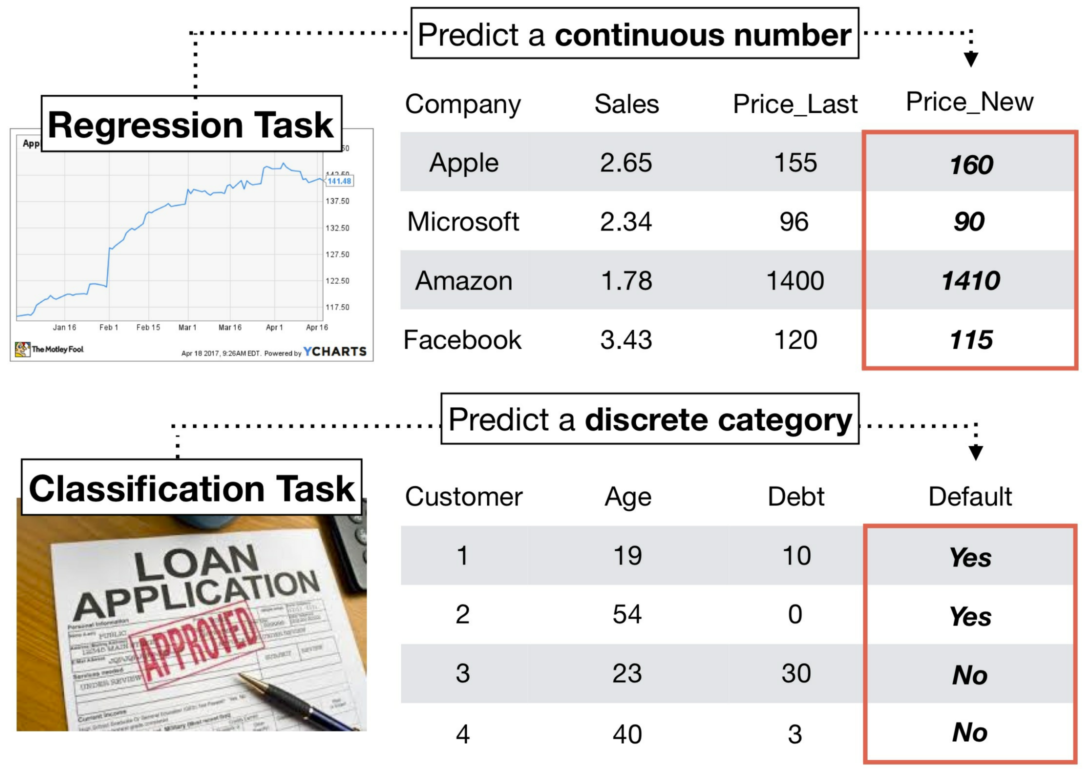
Source: Medium.com

What type of task do you have?

There are **many types of of ML tasks**.

In this course, we will focus on 2 of the most popular

Type	Description	Example
Regression (supervised)	Predicting a number	Stock prices
Classification (supervised)	Predicting a category, like whether	Whether someone will purchase a product or not


Predict a **continuous number**
Regression Task
Predict a **discrete category**
Classification Task
LOAN APPLICATION
APPROVED

Company	Sales	Price_Last	Price_New
Apple	2.65	155	***160***
Microsoft	2.34	96	***90***
Amazon	1.78	1400	***1410***
Facebook	3.43	120	***115***

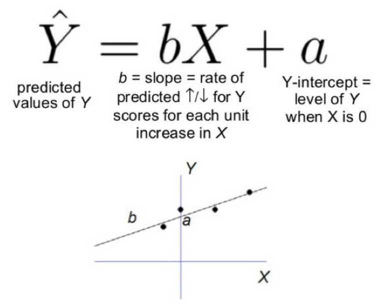
Customer	Age	Debt	Default
1	19	10	***Yes***
2	54	0	***Yes***
3	23	30	***No***
4	40	3	***No***

What ML models are there?

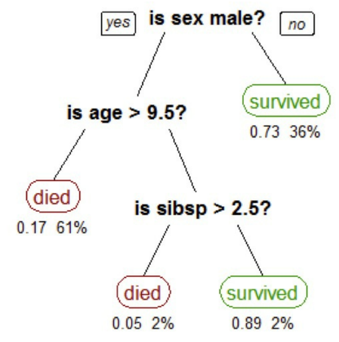
There are of machine learning models

In this course, you will learn 3 of the most popular:

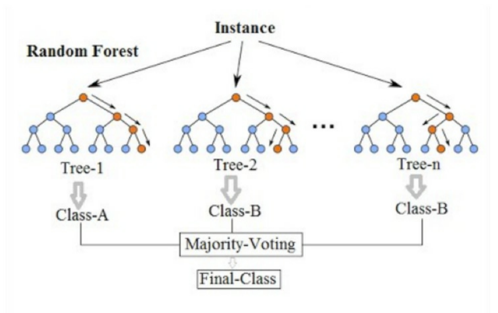
Model	Description
Regression	A weighted linear combination of features and weights
Decision Tree	A series of hierarchical 'yes/no' decisions
Random Forests	Combination of many decision trees


$\hat{Y} = bX + a$
predicted values of Y
b = slope = rate of predicted ↑/↓ for Y scores for each unit increase in X
Y-intercept = level of Y when X is 0
Y
b
a
X

Regression



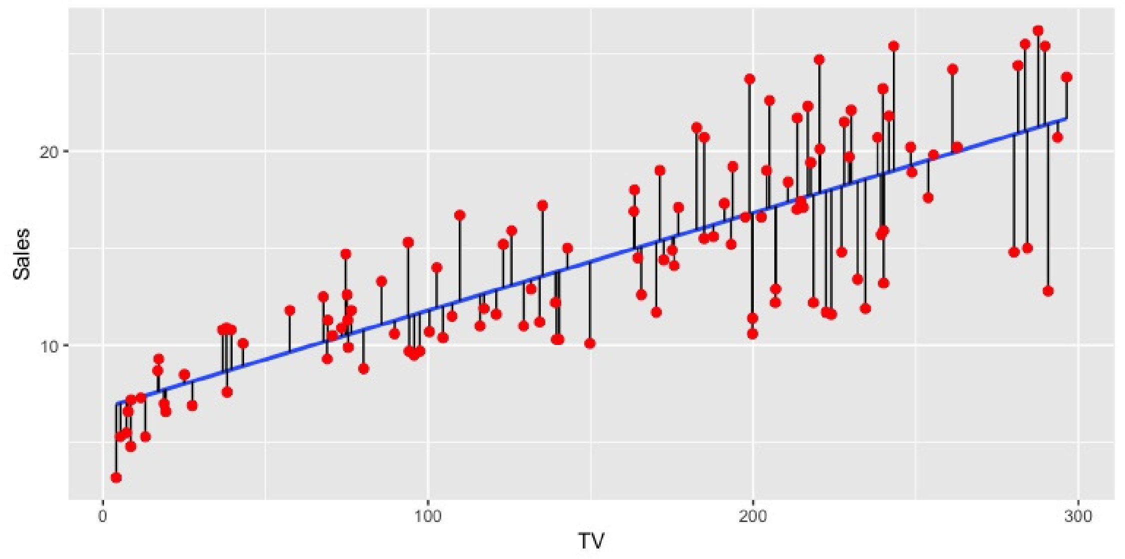
Decision Tree



Random Forest

Once you have a model you need to "Fit" (aka, "Train") it to data.

What does that mean?



James et al., Introduction to SL

What does "Fitting" mean?

For any model class, there is no "One" single model.

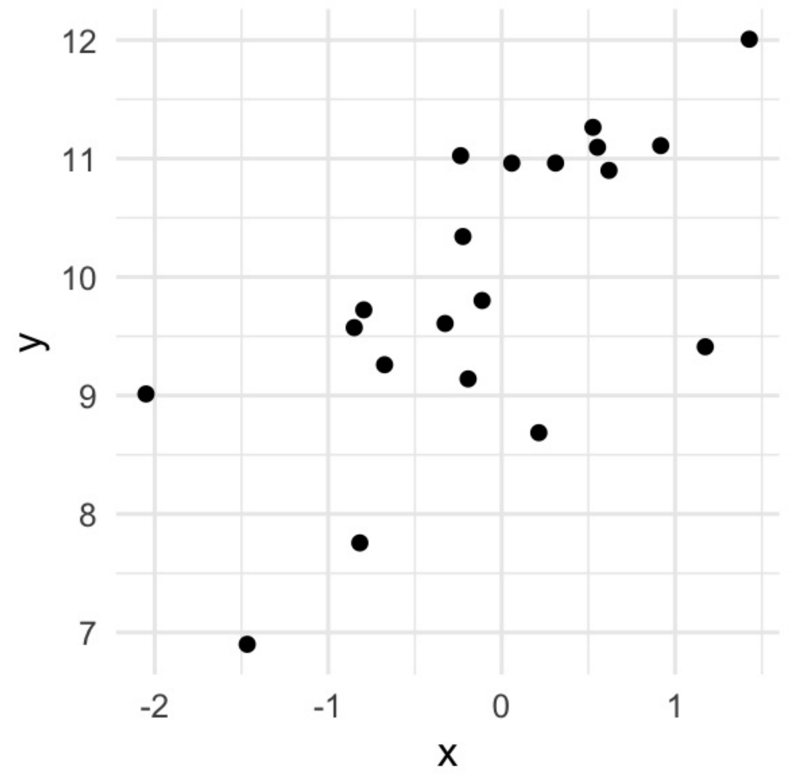
> How many possible Regression models are there?

Any machine learning model can be 'fit' (aka 'trained') on a specific dataset of interest. This means **finding the "best" version of a model** for a specific dataset.

> "Let me represent the data in the best way I can given how I work"
> ~ Model during fitting

The "best" model is usually defined as a combination of **accuracy** (higher better!) and **complexity** (simpler is better!)

How do I fit a model to these data?



Defining Accuracy (or Error)

To train (fit) a model to a dataset, we need to **mathematically define Accuracy**

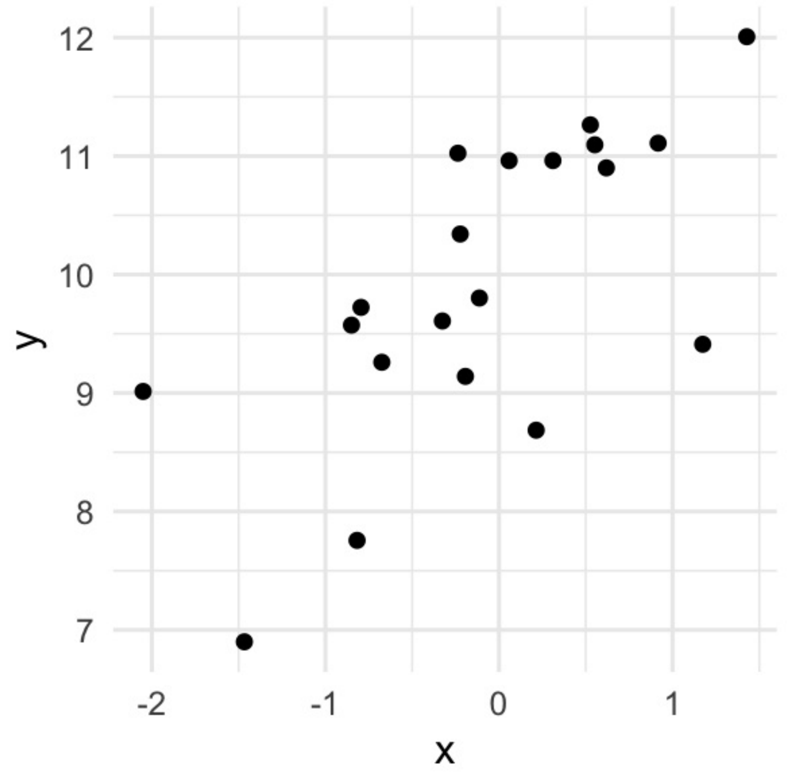
Alternatively, we can define a model's **Error**

There is **no 'correct'** definition of error, it depends on **what's important to you** as the decision maker!

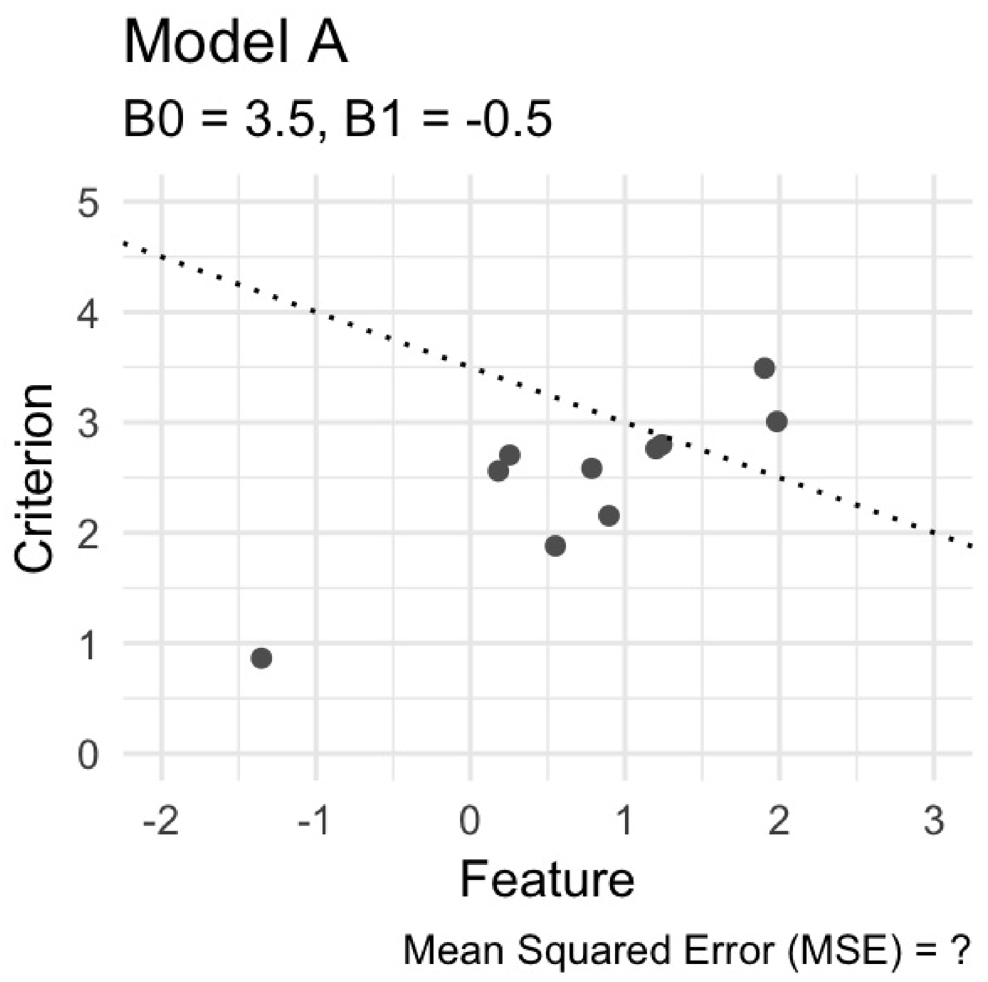
Once accuracy (or error) is defined, a model can be trained to maximize (or minimize) it!

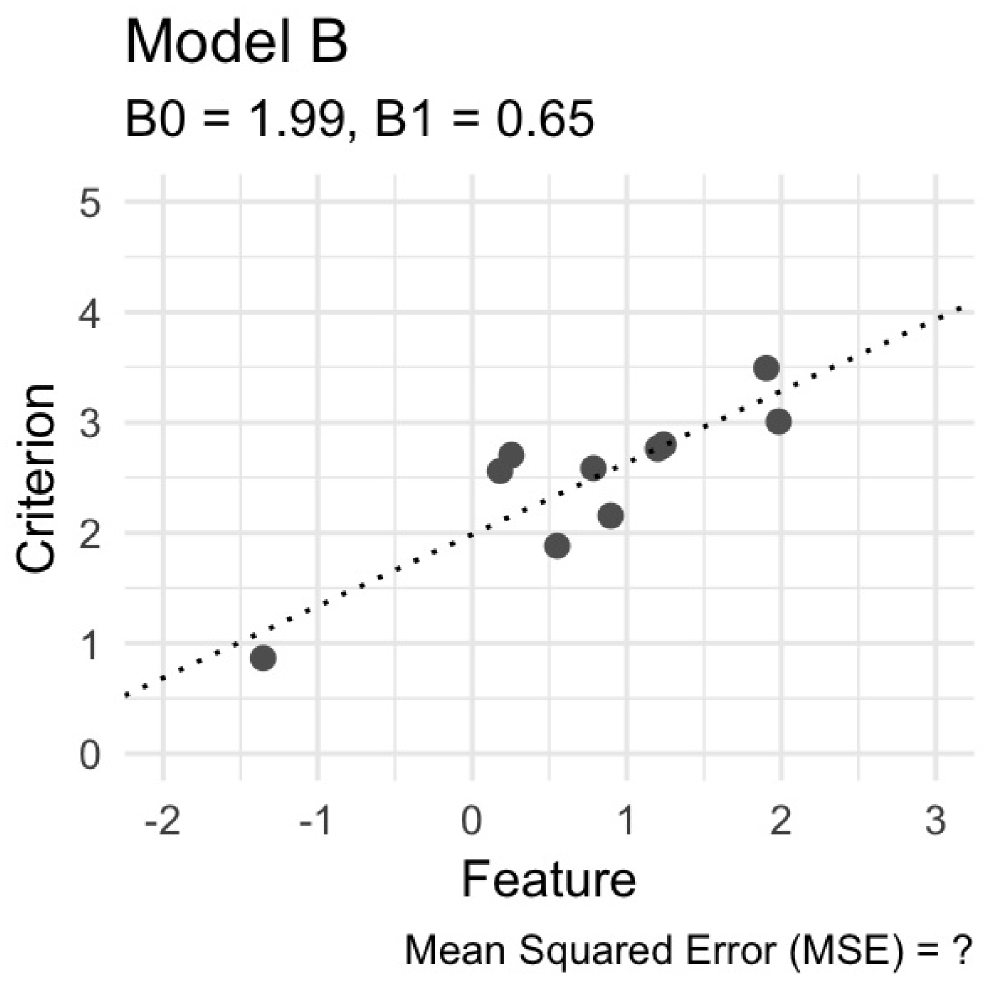
The model that minimizes error (or maximizes accuracy) is the final **Training model**

How do I fit a model to these data?

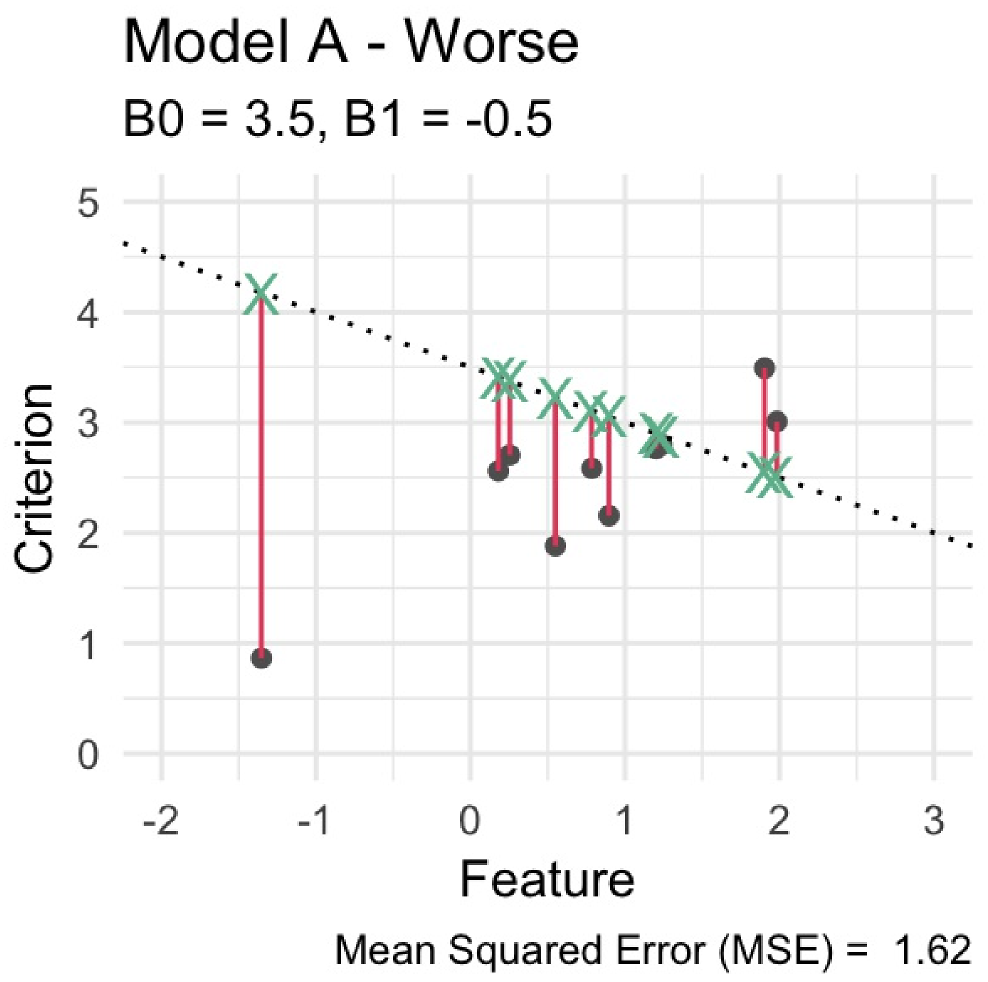


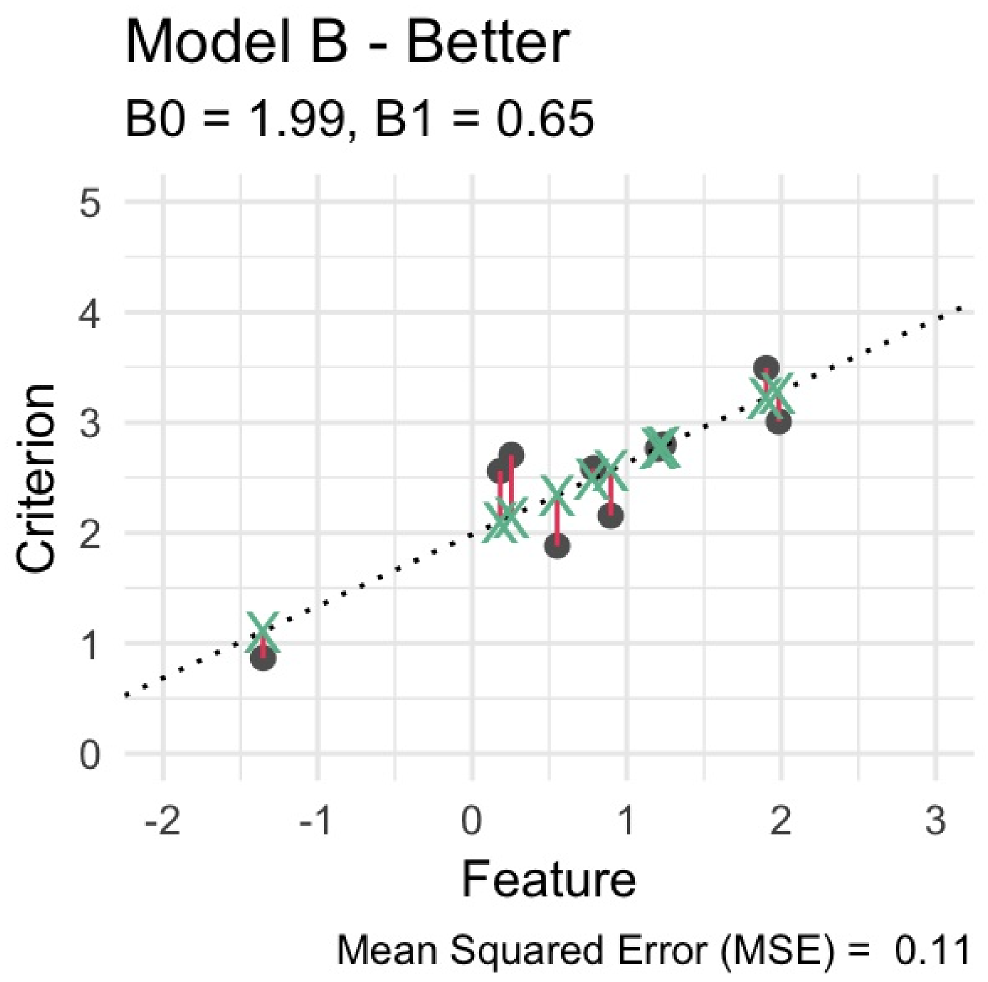
Which of these models is better? Why?





Which of these models is better? Why?





Regression Error

MAE: Mean Absolute Error

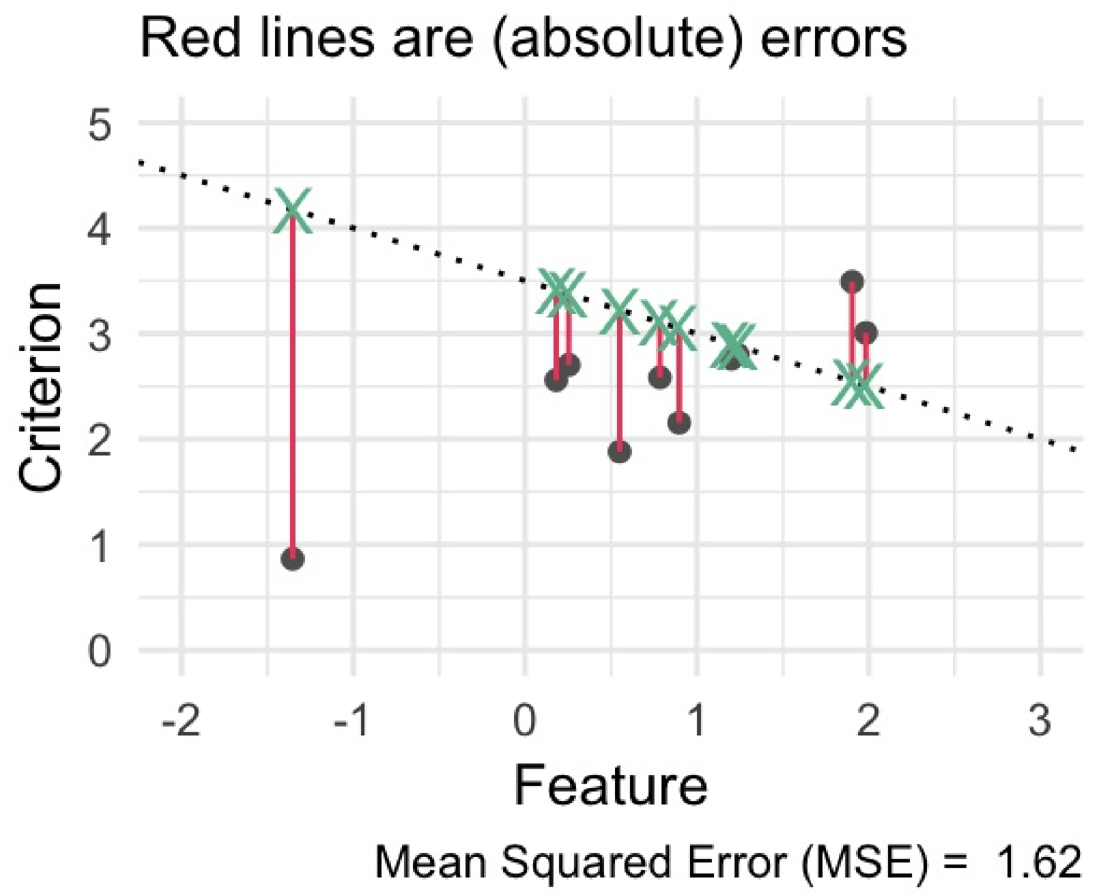
$$\large MAE = \frac{1}{n}\sum_{i=1}^{n} \lvert Prediction_{i} - Truth_{i} \rvert$$

> On average, how far are predictions away from true values?

MSE: Mean Squared Error

$$\large MSE = \frac{1}{n}\sum_{i=1}^{n} (Prediction_{i} - Truth_{i})^{2}$$

> On average, how far are predictions away from true values (squared!)?



Classification Accuracy

Classification accuracy measures all come from the **"confusion matrix"**

The confusion matrix is a cross tabulation table showing predictions versus true classes.

Confusion Matrix

	Y is Positive	Y is Negative
Predict "Positive"	TP True Positive	FP False Positive
Predict "Negative"	FN False Negative	TN True Negative

Data

	X1	X2	X3	Prediction	Truth	Outcome
1	.	.	.	"Default"	Default	TP
2	.	.	.	"Default"	Default	TP
3	.	.	.	"Repay"	Repay	TN
4	.	.	.	"Default"	Repay	FP
5	.	.	.	"Repay"	Default	FN
6	.	.	.	"Default"	Default	TP
7	.	.	.	"Repay"	Repay	TN

Confusion Matrix

	True Default	True Repay
"Default"	3	1
"Repay"	1	2

Classification Accuracy

Classification accuracy measures all come from the **"confusion matrix"**

The confusion matrix is a cross tabulation table showing predictions versus true classes.

Confusion Matrix

	Y is Positive	Y is Negative
Predict "Positive"	TP True Positive	FP False Positive
Predict "Negative"	FN False Negative	TN True Negative

Overall Accuracy

> What percent of my predictions are correct?

$$\large Overall \; Accuracy = \frac{TP + TN}{ TP + TN + FN + FP}$$

Sensitivity

> , what percent of predictions are correct?

$$\large Sensitivity = \frac{TP}{ TP +FN }$$

Specificity

> , what percent of predictions are correct?

$$\large Specificity = \frac{TN}{ TN + FP }$$

Classification Accuracy

Example: Loan default

Imagine we use a model (e.g. a decision tree) to predict whether or not each of 7 customers will default on their loan.

After the loan period is over, we obtain the final confusion matrix comparing our predictions to the truth:

Confusion Matrix

	True Default	**True Repay**
Predict "Default"	TP 3	FP 1
Predict "Repay"	FN 1	TN 2

Overall Accuracy

Across all customers, our model has an accuracy of 71%

$$\large Overall \; Accuracy = \frac{3 + 2}{3 + 2 + 1 + 1} = 0.71$$

Sensitivity

Our model is 75% accurate in catching true defaults

$$\large Sensitivity = \frac{3}{3 + 4} = .75$$

Specificity

Our model is 67% accurate in catching true repayments

$$\large Specificity = \frac{2}{ 2 + 1 }= 0.67$$

Ready to fit!

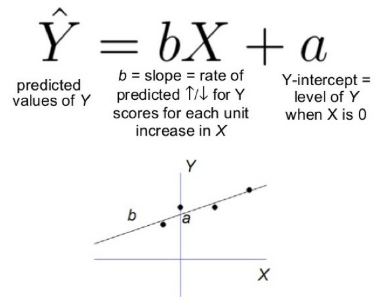
Now we're ready to fit models to data!

In this course will cover three commonly used models, **Regression**, **Decision Trees**, and **Random Forest**.

These models can be used in both regression and classification tasks.

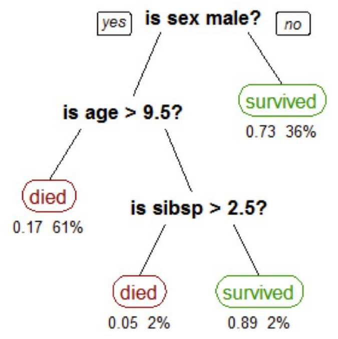
As you'll see, they differ in complexity in important regards.

Model	Complexity
Regression	Medium
Decision Tree	Low (usually)
Random Forests	High

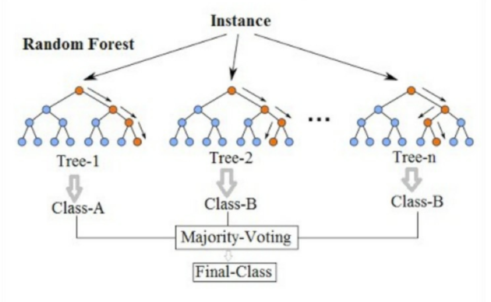


$\hat{Y} = bX + a$
predicted values of Y
b = slope = rate of predicted ↑/↓ for Y scores for each unit increase in X
Y-intercept = level of Y when X is 0
Y
b
a
X

Regression

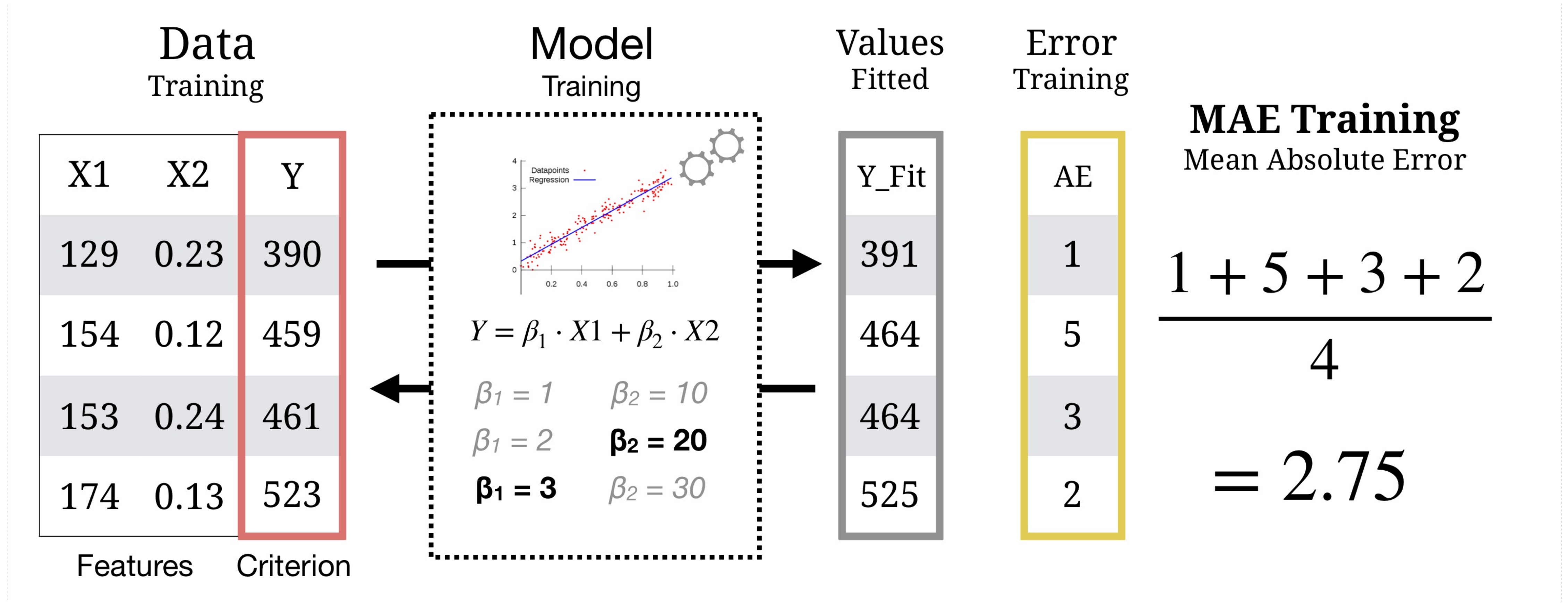


Decision Tree



Random Forest

Model Training (aka fitting)



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

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

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

Y_Fit	AE
391	1
464	5
464	3
525	2

MAE Training

Mean Absolute Error

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

$$= 2.75$$

Regression

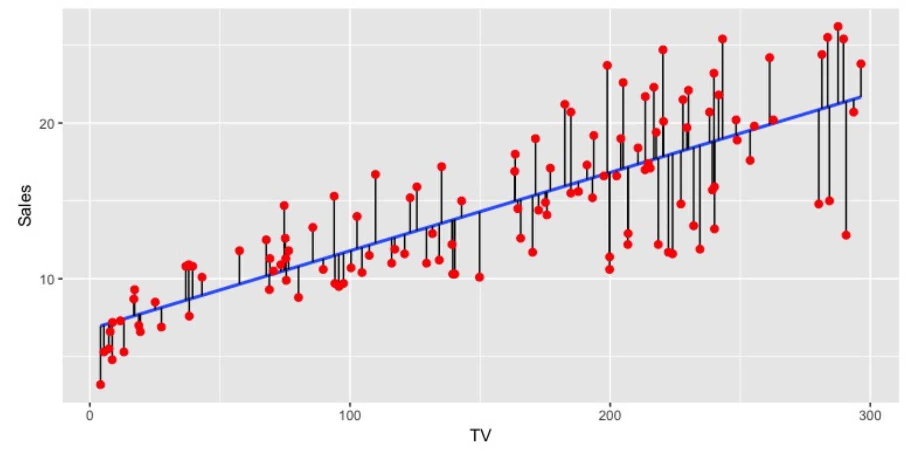
Decision Trees

Random Forests

Regression

In **regression**, the criterion Y is modeled as the **sum** of **predictors times weights** β_{1}, β_{2}.

$$\hat{Y} = \beta_{0} + X1 \times \beta_{X1} + X2 \times \beta_{X2} + ...$$



James et al., Introduction to SL

Interpretation

$$\LARGE \hat{Y} = \beta_{0} + X1 \times \beta_{X1} + X2 \times \beta_{X2} + ...$$

Each beta weight β_{i} can be interpreted as:

"As the value of X_{i} increases by 1, how does the criterion Y change?"

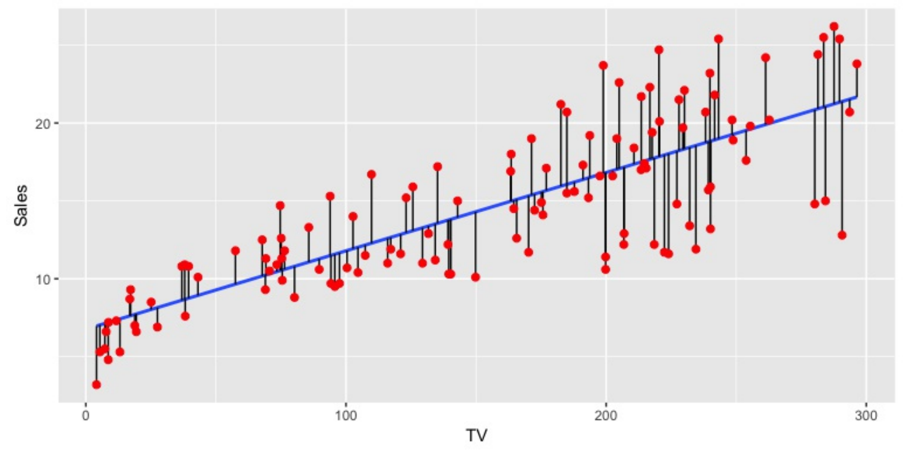
The more extreme β_{i} is (either positive or negative), the more X_{i} is used to predict the criterion Y (Note: take into account the scale of X_{i}!).

If a value of β_{i} is exactly 0, that means X_{i} does not help us predict the criterion Y

Regression

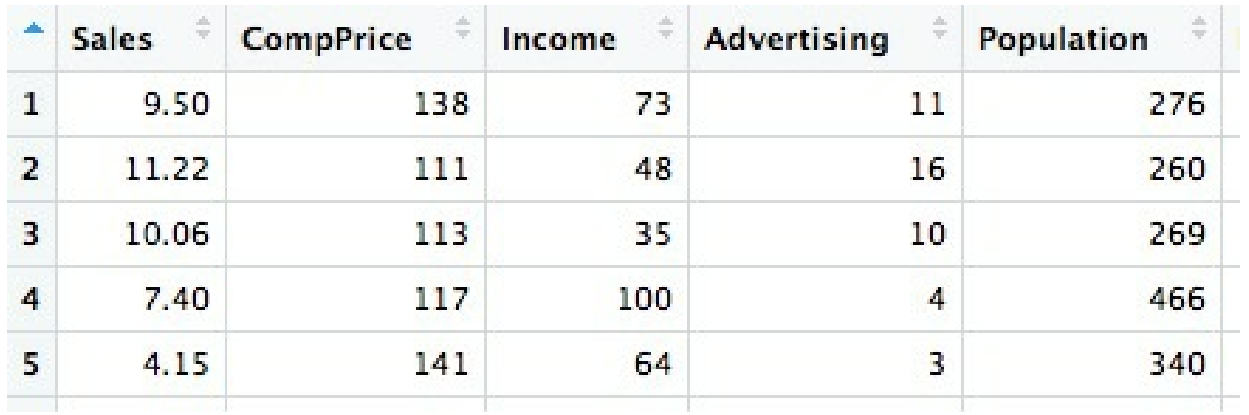
In **regression**, the criterion Y is modeled as the **sum** of **predictors times weights** \ (\beta_{1}\), \(\beta_{2}\).

$$\hat{Y} = \beta_{0} + X1 \times \beta_{X1} + X2 \times \beta_{X2} + ...$$



James et al., Introduction to SL

Sales Example



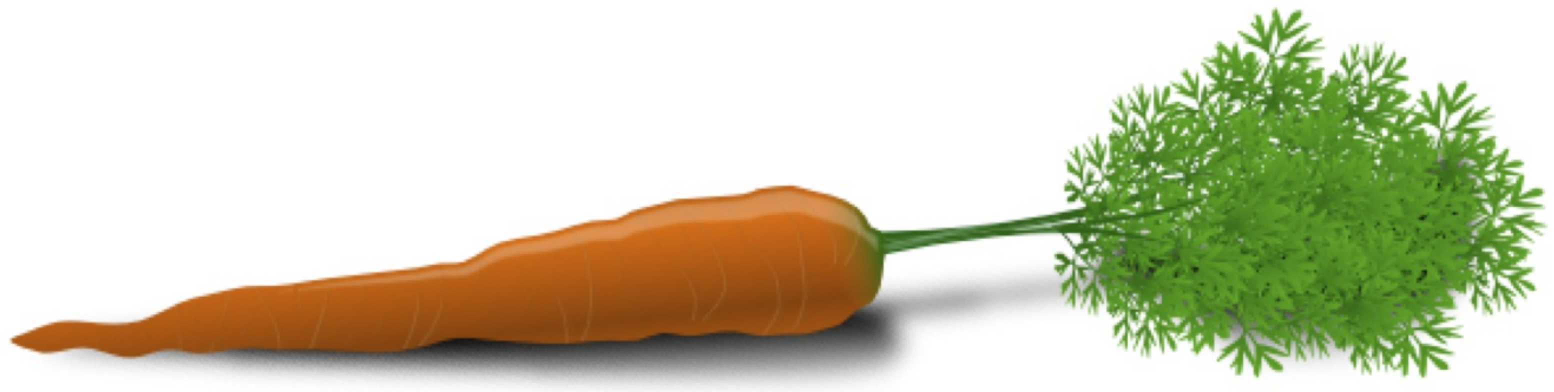
	Sales	CompPrice	Income	Advertising	Population
1	9.50	138	73	11	276
2	11.22	111	48	16	260
3	10.06	113	35	10	269
4	7.40	117	100	4	466
5	4.15	141	64	3	340

Regression Model

$$\large Sales = \beta_{0} + CompPrice \times \beta_{CompPrice} + Income \times \beta_{Income} + ...$$ Estimates

$$\large Sales = 10 + CompPrice \times 5.4 + Income \times 1.3 + ...$$

Let's fit regression models with caret!



caret

Main caret fitting functions

Function	Purpose
trainControl()	Determine how training (in general) will be done
train()	Specify a model and find parameters
predict()	Predict values (either fitted values or predictions for new data)
postResample()	Evaluate model performance (fitting or prediction)

```
# Step 1: Load data
#   read_csv()

data_train <- read_csv(...)

# Step 2: Define control parameters
#   trainControl()

ctrl <- trainControl(...)

# Step 3: Train and explore model
#   train()

mod <- train(...)
summary(mod)
mod$finalModel    # see final model

# Step 4: Assess fit
#   predict(), postResample()

fit <- predict()
postResample(fit, truth)

# Step 5: Visualise results

ggplot(...)
```

trainControl()

Use `trainControl()` to define how `caret` should, generally, **select the best parameters** for an ML model.

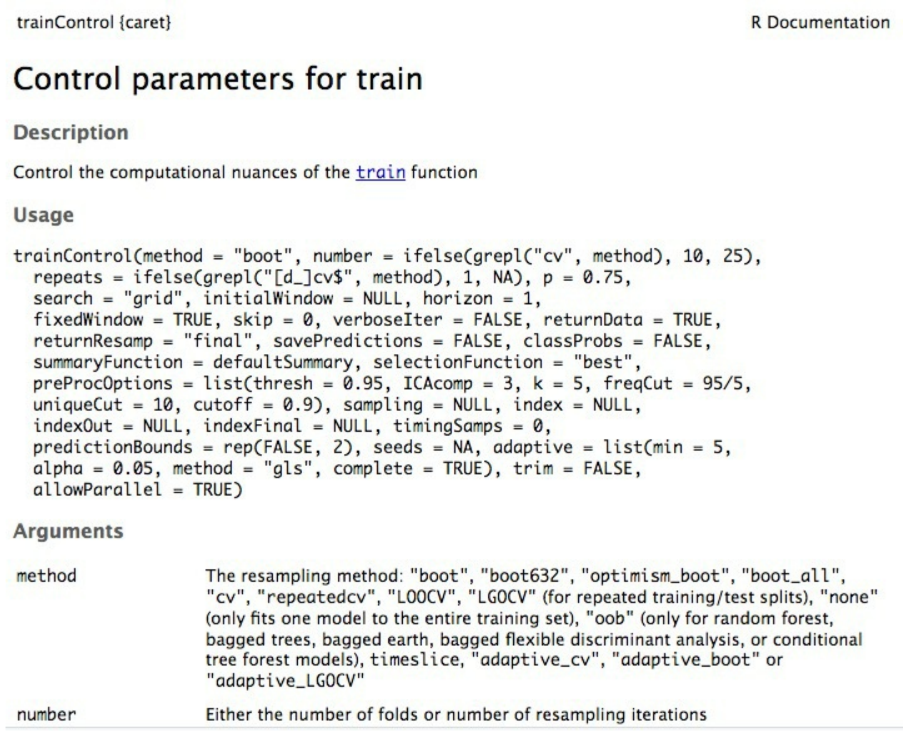
Here you can tell `caret` to do things like repeated **cross validation** (which we will learn about later).

Argument	Description
`method`	How should fitting be done?

For now, we'll set `method = "none"` to keep things simple to **fit the model without advanced parameter tuning**.

```r
# Fit the model without any
#  advanced parameter tuning methods

ctrl <- trainControl(method = "none")
```

```r
?trainControl
```



trainControl {caret} R Documentation

Control parameters for train

Description

Control the computational nuances of the train function

Usage

```r
trainControl(method = "boot", number = ifelse(grepl("cv", method), 10, 25),
  repeats = ifelse(grepl("[d_]cv$", method), 1, NA), p = 0.75,
  search = "grid", initialWindow = NULL, horizon = 1,
  fixedWindow = TRUE, skip = 0, verboseIter = FALSE, returnData = TRUE,
  returnResamp = "final", savePredictions = FALSE, classProbs = FALSE,
  summaryFunction = defaultSummary, selectionFunction = "best",
  preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5, freqCut = 95/5,
  uniqueCut = 10, cutoff = 0.9), sampling = NULL, index = NULL,
  indexOut = NULL, indexFinal = NULL, timingSamps = 0,
  predictionBounds = rep(FALSE, 2), seeds = NA, adaptive = list(min = 5,
  alpha = 0.05, method = "gls", complete = TRUE), trim = FALSE,
  allowParallel = TRUE)
```

Arguments

method	The resampling method: "boot", "boot632", "optimism_boot", "boot_all", "cv", "repeatedcv", "LOOCV", "LGOCV" (for repeated training/test splits), "none" (only fits one model to the entire training set), "oob" (only for random forest, bagged trees, bagged earth, bagged flexible discriminant analysis, or conditional tree forest models), timeslice, "adaptive_cv", "adaptive_boot" or "adaptive_LGOCV"
number	Either the number of folds or number of resampling iterations

train()

train() is the workhorse fitting function of caret.

With just this one function, you can **fit any of 200+ models** just by changing the **method** argument!

Argument	Description
form	Formula specifying criterion
data	Training data
method	Model
trControl	Control parameters

Train a Regression model

Regression: `method = "glm"`

```
# Fit a regression model predicting Price

mod <- train(form = income ~ ., # Formula
             data = baselers,   # Training data
             method = "glm",    # Regression
             trControl = ctrl)  # Control Parameters
```

train()

train() is the workhorse fitting function of caret.

With just this one function, you can **fit any of 200+ models** just by changing the **method** argument!

Argument	Description
form	Formula specifying criterion
data	Training data
method	Model
trControl	Control parameters

Train a Random Forest model

Random Forest: method = "rf"

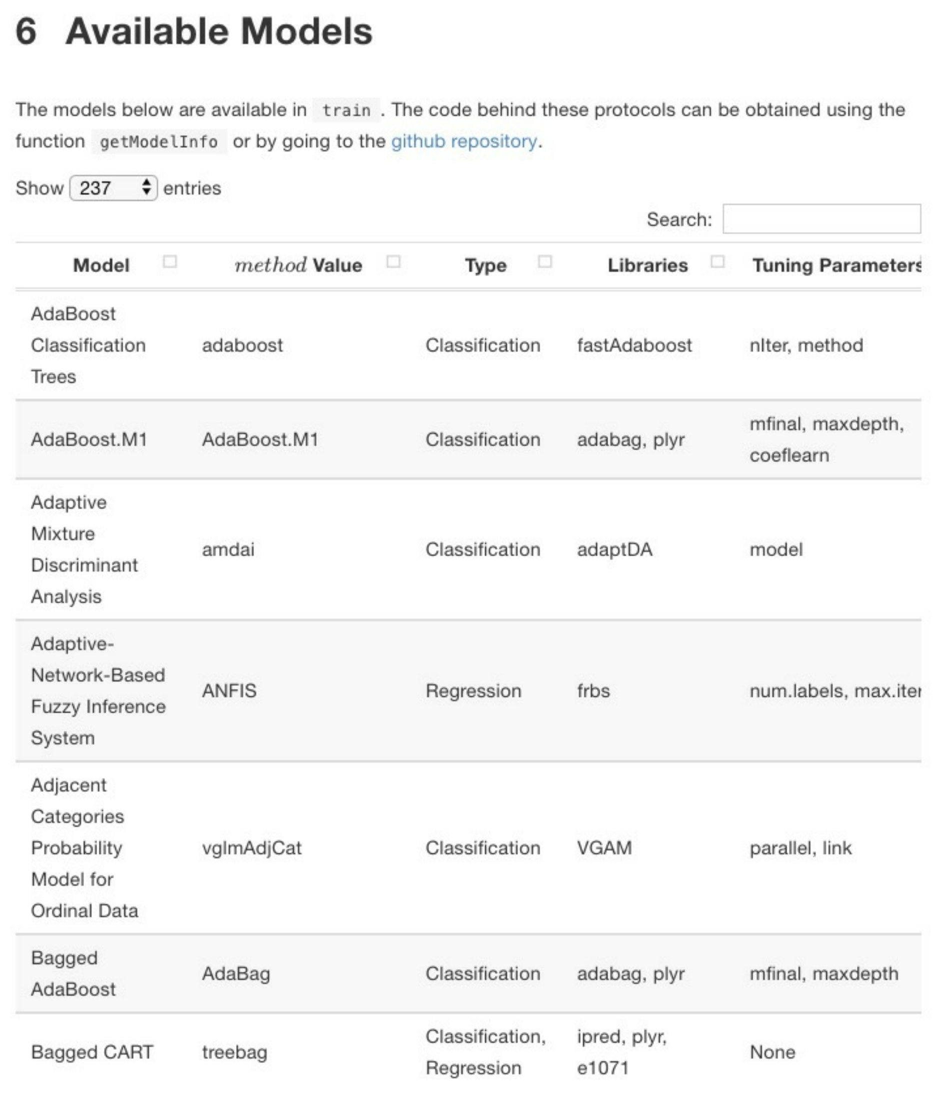
```
# Fit a Random Forest model predicting Price

mod <- train(form = income ~ ., # Formula
             data = baselers,   # Training data
             method = "rf",     # Random Forests
             trControl = ctrl)  # Control Parameters
```

train()

`train()` is the workhorse fitting function of `caret`.

With just this one function, you can **fit any of 200+ models** just by changing the **method** argument!

Find all 280+ models **here**.



6 Available Models

The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 237 entries

Search:

Model	*method* Value	Type	Libraries	Tuning Parameters
AdaBoost Classification Trees	adaboost	Classification	fastAdaboost	nIter, method
AdaBoost.M1	AdaBoost.M1	Classification	adabag, plyr	mfinal, maxdepth, coeflearn
Adaptive Mixture Discriminant Analysis	amdai	Classification	adaptDA	model
Adaptive-Network-Based Fuzzy Inference System	ANFIS	Regression	frbs	num.labels, max.iter
Adjacent Categories Probability Model for Ordinal Data	vglmAdjCat	Classification	VGAM	parallel, link
Bagged AdaBoost	AdaBag	Classification	adabag, plyr	mfinal, maxdepth
Bagged CART	treebag	Classification, Regression	ipred, plyr, e1071	None

train()

Make sure your criterion is the correct class for your type of modelling task

- Numeric criterion = Regression Task
- Factor criterion = Classification Task

```
# My training data
Loans
```

```
# A tibble: 5 x 5
  Default   Age Gender Cards Education
    <dbl> <dbl> <chr>  <dbl>     <dbl>
1       0    45 M          3        11
2       1    36 F          2        14
3       0    76 F          5        12
4       1    25 M          2        17
5       1    36 F          3        12
```

See that the column Default is 0's and 1's, but is coded as numeric.

This code will think that Default is a continuous number, not a category (probably not what you want)

```
# Will be a regression task if Default is numeric

mod <- train(form = default ~ .,
             data = Loans,
             method = "glm",
             trControl = ctrl)
```

Warning messages:...Are you sure you wanted to do regression?

Use `factor()` to **convert your criterion** to a factor, now you are doing classification!

```
# Will be a classification task

mod <- train(form = factor(Default) ~ .,
             data = Loans,
             method = "glm")
```

.$finalModel

The `train()` function returns a list with a key object called `finalModel` - this is your final machine learning model!

You can access the model with `mod$finalModel`, and explore the object with generic functions:

Function	Description
`summary()`	Overview of the most important information
`names()`	See all named elements you can access with $

```r
# Create a regression object
mod <- train(form = income ~ age + height + fitness,
             data = baselers)   # Training data

# Look at final model
mod$finalModel
# [...]
```

```r
# Look at all named outputs
names(mod$finalModel)
```

```
 [1] "coefficients"      "residuals"         "fitted.values"     "effe
 [6] "rank"              "qr"                "family"            "line
[11] "aic"               "null.deviance"     "iter"              "weig
[16] "df.residual"       "df.null"           "y"                 "conv
[21] "model"             "formula"           "terms"             "data
[26] "control"           "method"            "contrasts"         "xlev
[31] "problemType"       "tuneValue"         "obsLevels"         "para
```

```r
# Access specific outputs
mod$finalModel$coefficients
```

```
(Intercept)         age      height     fitness
    136.606     151.751       3.381      11.012
```

predict()

The `predict()` function is allows you to return predictions from a model.

Put your model object as the first argument. If you don't specify a new dataset with `newdata`, the function returns **fitted values from training**

```r
# Get fitted values
glm_fits <- predict(object = mod)
```

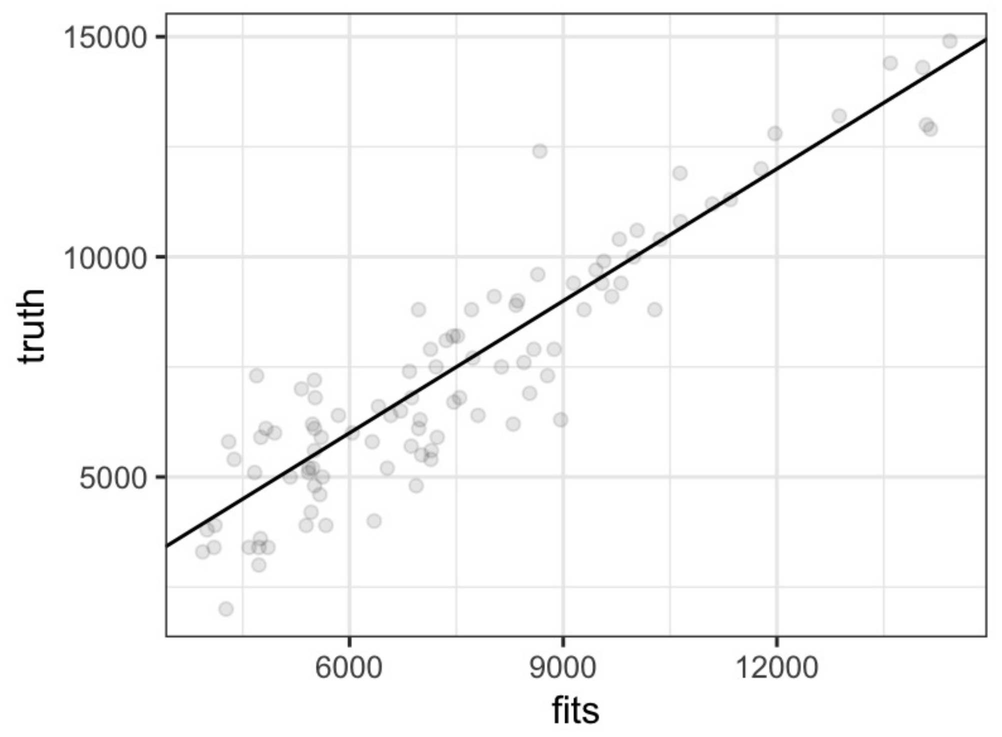
The result is a vector

```r
# Result is a vector of fits
glm_fits[1:5]
```

```
   1    2    3    4    5 
5507 6971 6969 8643 5324 
```

Plot of fits versus Truth

If the model was perfect, all points would be on diagonal



postResample()

To calculate aggregate model performance, use `postResample()`

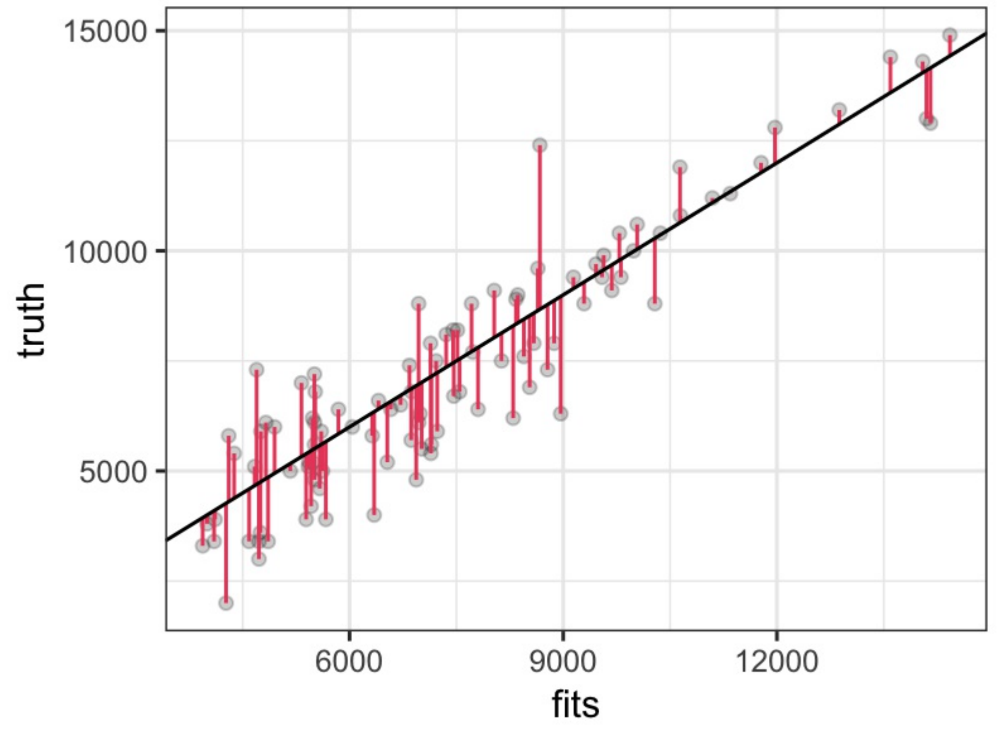
Argument	Description
pred	Model predictions (or fits)
obs	The observed (true) values

```
# Assess performance with postResample()

postResample(pred = glm_fits, # Predictions
             obs = baselers$income) # Truth
```

```
    RMSE  Rsquared       MAE
1172.905     0.821   936.994
```

Plot of fits versus Truth

Red lines indicate absolute error(s)



Questions?

Practical