MACHINE LEARNING TOOLBOX

Median imputation

Dealing with missing values

- Most models require numbers, can't handle missing data
- Common approach: remove rows with missing data
  - Can lead to biases in data
  - Generate over-confident models
- Better strategy: median imputation!
  - Replace missing values with medians
  - Works well if data *missing at random* (MAR)

Example: mtcars

```
# Generate some data with missing values
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA

# Split target from predictors
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Try to fit a caret model
> library(caret)
> model <- train(x = X, y = Y)
Error in train.default(x = X, y = Y) : Stopping
```

A simple solution

```
# Now fit with median imputation
> model <- train(x = X, y = Y, preProcess = "medianImpute")
> print(model)
Random Forest

32 samples
 3 predictor

Pre-processing: median imputation (3)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 32, 32, 32, 32, 32, 32, ...
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared
  2     2.617096  0.8234652
  3     2.670550  0.8164535

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 2.
```

Let's practice!

KNN imputation

Dealing with missing values

- Median imputation is fast, but...
- Can produce incorrect results if data *missing not at random*
- k-nearest neighbors (KNN) imputation
- Imputes based on "similar" non-missing rows

Example: missing not at random

- Pretend smaller cars don't report horsepower
- Median imputation incorrect in this case

Assumes small cars have medium-large horsepower

```
# Generate data with missing values
> data(mtcars)
> mtcars[mtcars$disp < 140, "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Use median imputation
> set.seed(42)
> model <- train(x = X, y = Y, method = "glm",
                 preProcess = "medianImpute")
> print(min(model$results$RMSE))
[1] 3.612713
```

Example: missing not at random

- KNN imputation is better
- Uses cars with similar `disp` / `cyl` to impute
- Yields a more accurate (but slower) model

```
# Use KNN imputation
> set.seed(42)
> model <- train(x = X, y = Y,
                 method = "glm",
                 preProcess = "knnImpute"
  )
> print(min(model$results$RMSE))
[1] 3.558881
```

Compare to 3.61 for median imputation

Let's practice!

Multiple preprocessing methods

The wide world of `preProcess`

- You can do a lot more than median or knn imputation!
- Can chain together multiple preprocessing steps
- Common "recipe" for linear models (order matters!)
  - **Median imputation -> center -> scale -> fit `glm`**
- See `?preProcess` for more detail

Example: preprocessing `mtcars`

```
# Generate some data with missing values
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[,2:4]

# Use linear model "recipe"
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale")
  )
> print(min(model$results$RMSE))
[1] 3.612713
```

Missing at random

Example: preprocessing `mtcars`

```
# PCA before modeling
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 3.402557
```

Example: preprocessing `mtcars`

```
# Spatial sign transform
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "spatialSign"))
> min(model$results$RMSE)
[1] 4.284904
```

Preprocessing cheat sheet

- Start with median imputation
  - **Try KNN imputation if data missing not at random**
- For linear models...
  - Center and scale
  - Try PCA and spatial sign
- Tree-based models don't need much preprocessing

Let's practice!

MACHINE LEARNING TOOLBOX

Handling low-information predictors

No (or low) variance variables

- Some variables don't contain much information
  - Constant (i.e. no variance)
  - Nearly constant (i.e. low variance)
- Easy for one fold of CV to end up with constant column
- Can cause problems for your models
- Usually remove extremely low variance variables

Example: constant column in `mtcars`

```
# Reproduce dataset from last video
> data(mtcars)
> set.seed(42)
> mtcars[sample(1:nrow(mtcars), 10), "hp"] <- NA
> Y <- mtcars$mpg
> X <- mtcars[, 2:4]

# Add constant-valued column to mtcars
> X$bad <- 1
```

Example: constant column in `mtcars`

```
# Try to fit a model with PCA + glm
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("medianImpute", "center", "scale", "pca")
  )

Warning in preProcess.default(thresh = 0.95, k = 5, method =
c("medianImpute",   :
  These variables have zero variances: bad
Something is wrong; all the RMSE metric values are missing:
      RMSE            Rsquared
 Min.   : NA     Min.   : NA
 1st Qu.: NA     1st Qu.: NA
 Median : NA     Median : NA
 Mean   :NaN     Mean   :NaN
 3rd Qu.: NA     3rd Qu.: NA
 Max.   : NA     Max.   : NA
 NA's   :1       NA's   :1
```

caret to the rescue (again)

- "zv" removes constant columns
- "nzv" removes nearly constant columns

```
# Have caret remove those columns during modeling
> set.seed(42)
> model <- train(
    x = X, y = Y, method = "glm",
    preProcess = c("zv", "medianImpute", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 3.402557
```

Let's practice!

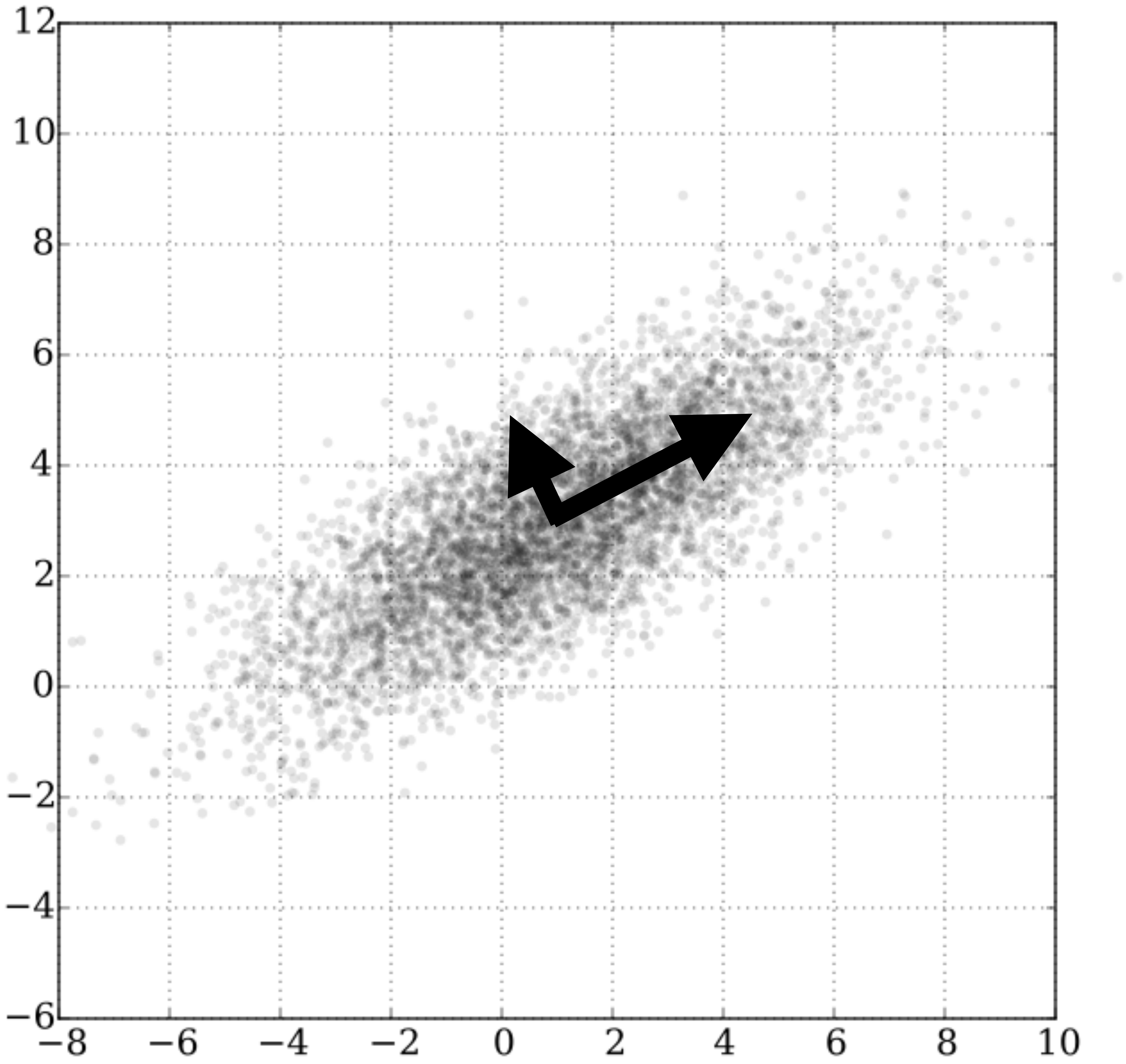
Principle components analysis (PCA)

Principle components analysis

- Combines low-variance and correlated variables
- Single set of high-variance, perpendicular predictors
- Prevents collinearity (i.e. correlation among predictors)

PCA: a visual representation

- First component has highest variance
- Second component has second highest variance
- And so on...



Example: blood-brain data

- Lots of predictors
- Many of them low-variance

```
# Load the blood brain dataset
> data(BloodBrain)
> names(bbbDescr)[nearZeroVar(bbbDescr)]
[1] "negative"      "peoe_vsa.2.1" "peoe_vsa.3.1" "a_acid"
[5] "vsa_acid"      "frac.anion7." "alert"
```

Example: blood-brain data

```
# Basic model
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("zv", "center", "scale")
  )
> min(model$results$RMSE)
[1] 1.107702
```

Example: blood-brain data

```
# Remove low-variance predictors
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("nzv", "center", "scale")
  )
> min(model$results$RMSE)
[1] 0.9796199
```

Example: blood-brain data

```
# Add PCA
> set.seed(42)
> data(BloodBrain)
> model <- train(
    x = bbbDescr, y = logBBB, method = "glm",
    trControl = trainControl(method = "cv", number = 10, verbose = TRUE),
    preProcess = c("zv", "center", "scale", "pca")
  )
> min(model$results$RMSE)
[1] 0.9796199
```

Let’s practice!