Supervised Machine Learning

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(parsnip)
```

Step 1: Collect Data We will use the iris dataset as an example.

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Step 2: Clean and Process Data

- Make sure data is tidy (rows are observations and columns are variables)
- Optionally, select only the columns of dataset that you are interested in using for the model
- Either remove rows with NA values or set to column mean

```
### Examples ###
## remove rows with NA values
noNAs <- filter(starwars, !is.na(mass))

## replace with mean
replaceWithMeans <- mutate(starwars,
                           mass = ifelse(is.na(mass),
                                         mean(mass),
                                         mass))
```

- Code categorical variables as numeric integers using `as.integer()` and `factor()`, if needed
  - If you are using the variable as the outcome for classification, only use `factor()`
  - If you want to use the variable as a feature, use `as.integer()`

```r
### Examples ###
## If categorical variable is already a factor
irisAllNumeric <- mutate(iris, Species = as.integer(Species))

## If categorical variable is character, need to make it a factor first
intSpecies <- starwars |>
  mutate(species = as.integer(factor(species)))
```

- Normalize numeric values using `scale()`, if needed

```r
### Example ###
irisNorm <- iris[,1:4] |>
  scale() |>
  as.data.frame()

irisNorm$Species <- irisAllNumeric$Species
```

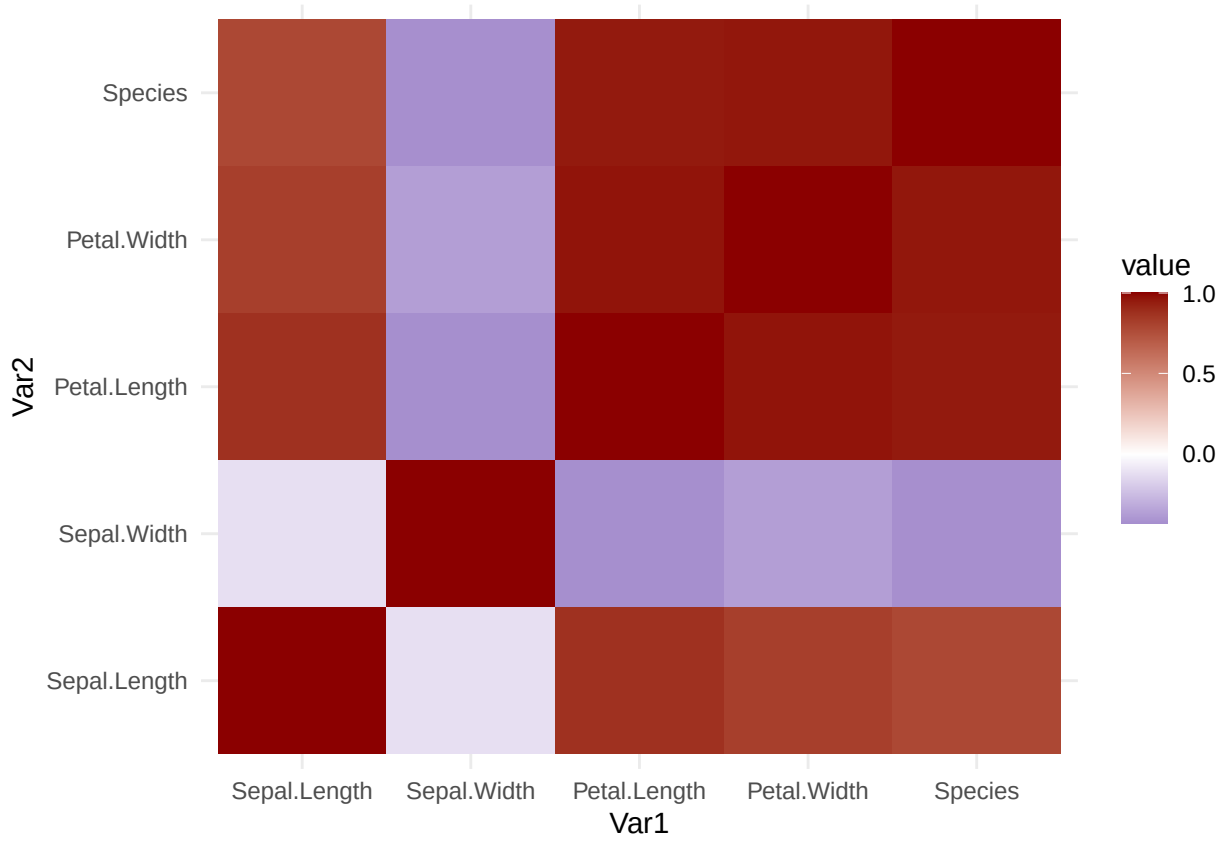
Step 3: Visualize Data

- Create scatterplots of different variables
- Use PCA to find variables with high variation
- Use `cor()` to determine correlation

```r
library(reshape2)
library(ggplot2)

## Calculate correlations
irisCorrelation <- cor(irisNorm) |>
  melt() |>
  as.data.frame()

## Plot correlations
ggplot(irisCorrelation, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "darkblue", mid = "white", high = "darkred", midpoint = 0) +
  theme_minimal()
```



Step 4: Perform Feature Selection

- Start with highly correlated variables
- Include variables that account for large amounts of variation

Step 5: Separate Data into Testing and Training Sets

- Choose 70-85% of data to put in the training set

```
library(rsample)
## Sample subsets for regression modeling
# Set seed to make random sampling reproducable
set.seed(479)

# Put 75% of the data into the training set
data_reg_split <- initial_split(irisAllNumeric, prop = 0.75)

# Create data frames for the two sets:
train_reg_data <- training(data_reg_split)
test_reg_data  <- testing(data_reg_split)

## Sample subsets for classification modeling
# Put 75% of the data into the training set
```

```r
data_class_split <- initial_split(iris, prop = 0.75)

# Create data frames for the two sets:
train_class_data <- training(data_class_split)
test_class_data  <- testing(data_class_split)
```

Step 6: Choose Suitable Model

Model Name	Function	Engine	Mode
Linear Regression	`linear_reg()`	"lm" or "glm"	Regression
Logistic Regression	`logistic_reg()`	"glm"	Classification
Boosted Decision Trees	`boost_tree()`	"xgboost"	Regression or Classification
Random Forest	`rand_forest()`	"randomForest"	Regression or Classification

See more at https://parsnip.tidymodels.org/articles/Examples.html

Step 7: Train Model on Training Set Linear Regression

```r
## Linear model
linreg_fit <- linear_reg() |>
  set_engine("lm") |>
  set_mode("regression") |>
  fit(Petal.Length ~ ., data = train_reg_data)

summary(linreg_fit$fit)
```

```
##
## Call:
## stats::lm(formula = Petal.Length ~ ., data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9189 -0.1855 -0.0287  0.1730  0.6731
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.89401    0.35226  -2.538   0.0126 *
## Sepal.Length  0.76954    0.06708  11.473  < 2e-16 ***
## Sepal.Width  -0.62082    0.08635  -7.189 9.24e-11 ***
## Petal.Width   0.87424    0.13783   6.343 5.52e-09 ***
## Species       0.50564    0.12361   4.091 8.36e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.306 on 107 degrees of freedom
## Multiple R-squared:  0.9723, Adjusted R-squared:  0.9713
## F-statistic: 940.2 on 4 and 107 DF,  p-value: < 2.2e-16
```

Logistic Regression

```
## Logistic model
# filter to just 2 outcomes
binary_train_data <- filter(train_class_data, Species %in% c("setosa","virginica"))
binary_test_data <- filter(test_class_data, Species %in% c("setosa","virginica"))

logreg_fit <- logistic_reg() |>
  set_engine("glm") |>
  set_mode("classification") |>
  fit(Species ~ ., data = binary_train_data)
```

```
## Warning: ! Logistic regression is intended for modeling binary outcomes, but there are 3
##   levels in the outcome.
## i If this is unintended, adjust outcome levels accordingly or see the
##   `multinom_reg()` function.

## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logreg_fit$fit)
```

```
##
## Call:
## stats::glm(formula = Species ~ ., family = stats::binomial, data = data)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -22.019 856457.330       0        1
## Sepal.Length     -3.451 225025.783       0        1
## Sepal.Width      -2.646 137444.721       0        1
## Petal.Length     12.715 136055.645       0        1
## Petal.Width       7.358 222080.805       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.0813e+02  on 77  degrees of freedom
## Residual deviance: 6.5294e-10  on 73  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

Boosting Decision Tree

```
## Use for regression
boost_tree_fit <- boost_tree() |>
  set_engine("xgboost") |>
  set_mode("regression") |>
  fit(Sepal.Length ~ ., data = train_reg_data)

boost_tree_fit$fit
```

```
## ##### xgb.Booster
## raw: 25.7 Kb
## call:
##   xgboost::xgb.train(params = list(eta = 0.3, max_depth = 6, gamma = 0,
##     colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
##     subsample = 1), data = x$data, nrounds = 15, watchlist = x$watchlist,
##     verbose = 0, nthread = 1, objective = "reg:squarederror")
## params (as set within xgb.train):
##   eta = "0.3", max_depth = "6", gamma = "0", colsample_bytree = "1", colsample_bynode = "1", min_chi
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 4
## niter: 15
## nfeatures : 4
## evaluation_log:
##     iter training_rmse
##        1     3.8123328
##        2     2.7110208
## ---
##       14     0.1697425
##       15     0.1595608
```

```
## Use for classification
boost_tree_fit2 <- boost_tree() |>
  set_engine("xgboost") |>
  set_mode("classification") |>
  fit(Species ~., data = train_class_data)

boost_tree_fit2$fit
```

```
## ##### xgb.Booster
## raw: 42.2 Kb
## call:
##   xgboost::xgb.train(params = list(eta = 0.3, max_depth = 6, gamma = 0,
##     colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 1,
##     subsample = 1), data = x$data, nrounds = 15, watchlist = x$watchlist,
##     verbose = 0, nthread = 1, objective = "multi:softprob", num_class = 3L)
## params (as set within xgb.train):
##   eta = "0.3", max_depth = "6", gamma = "0", colsample_bytree = "1", colsample_bynode = "1", min_chi
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 4
## niter: 15
## nfeatures : 4
## evaluation_log:
##     iter training_mlogloss
##        1        0.73655668
##        2        0.52454089
## ---
##       14        0.04167337
```

```
##       15        0.03748201
```

Random Forest

```
## Use for regression
rand_forest_fit <- rand_forest() |>
  set_engine("ranger") |>
  set_mode("regression") |>
  fit(Sepal.Length ~ ., data = train_reg_data)

rand_forest_fit$fit
```

```
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.threads = 1,      verbose = FALSE, seed = sample
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      112
## Number of independent variables:  4
## Mtry:                             2
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1125685
## R squared (OOB):                  0.8459725
```

```
## Use for classification
rand_forest_fit2 <- rand_forest() |>
  set_engine("ranger") |>
  set_mode("classification") |>
  fit(Species ~., data = train_class_data)

rand_forest_fit2$fit
```

```
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.threads = 1,      verbose = FALSE, seed = sample
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      112
## Number of independent variables:  4
## Mtry:                             2
## Target node size:                 10
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.02756524
```

Step 8: Evaluate Performance on Test Dataset Use `predict()` with any model to predict the dependent variable.

```
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':
##
##     Recall
```

```
## Linear Regression
irisPred <- test_reg_data
irisPred$linReg <- predict(linreg_fit, test_reg_data)$.pred

# error
yardstick::mae(irisPred, truth = Petal.Length, estimate = linReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mae     standard       0.238
```

```
yardstick::rmse(irisPred, truth = Petal.Length, estimate = linReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.308
```

```
## Logistic Regression
irisPred <- binary_test_data
irisPred$logReg <- predict(logreg_fit, binary_test_data)$.pred_class

#f1 score
F1_Score(irisPred$Species, irisPred$logReg)
```

```
## [1] 1
```

```
## Boosted Decision Trees
# Regression
irisPred <- test_reg_data
irisPred$logReg <- predict(boost_tree_fit, test_reg_data)$.pred

# error
yardstick::mae(irisPred, truth = Sepal.Length, estimate = logReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mae     standard       0.296
```

```
yardstick::rmse(irisPred, truth = Sepal.Length, estimate = logReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.361
```

```
# Classification
irisPred <- test_class_data
irisPred$logReg <- predict(boost_tree_fit2, test_class_data)$.pred_class

#f1
F1_Score(irisPred$logReg, irisPred$Species)
```

```
## [1] 1
```

```
## Random Forest
# Regression
irisPred <- test_reg_data
irisPred$logReg <- predict(rand_forest_fit, test_reg_data)$.pred

# error
yardstick::mae(irisPred, truth = Sepal.Length, estimate = logReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mae     standard       0.308
```

```
yardstick::rmse(irisPred, truth = Sepal.Length, estimate = logReg)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.359
```

```
# Classification
irisPred <- test_class_data
irisPred$logReg <- predict(rand_forest_fit2, test_class_data)$.pred_class

#f1
F1_Score(irisPred$logReg, irisPred$Species)
```

```
## [1] 1
```