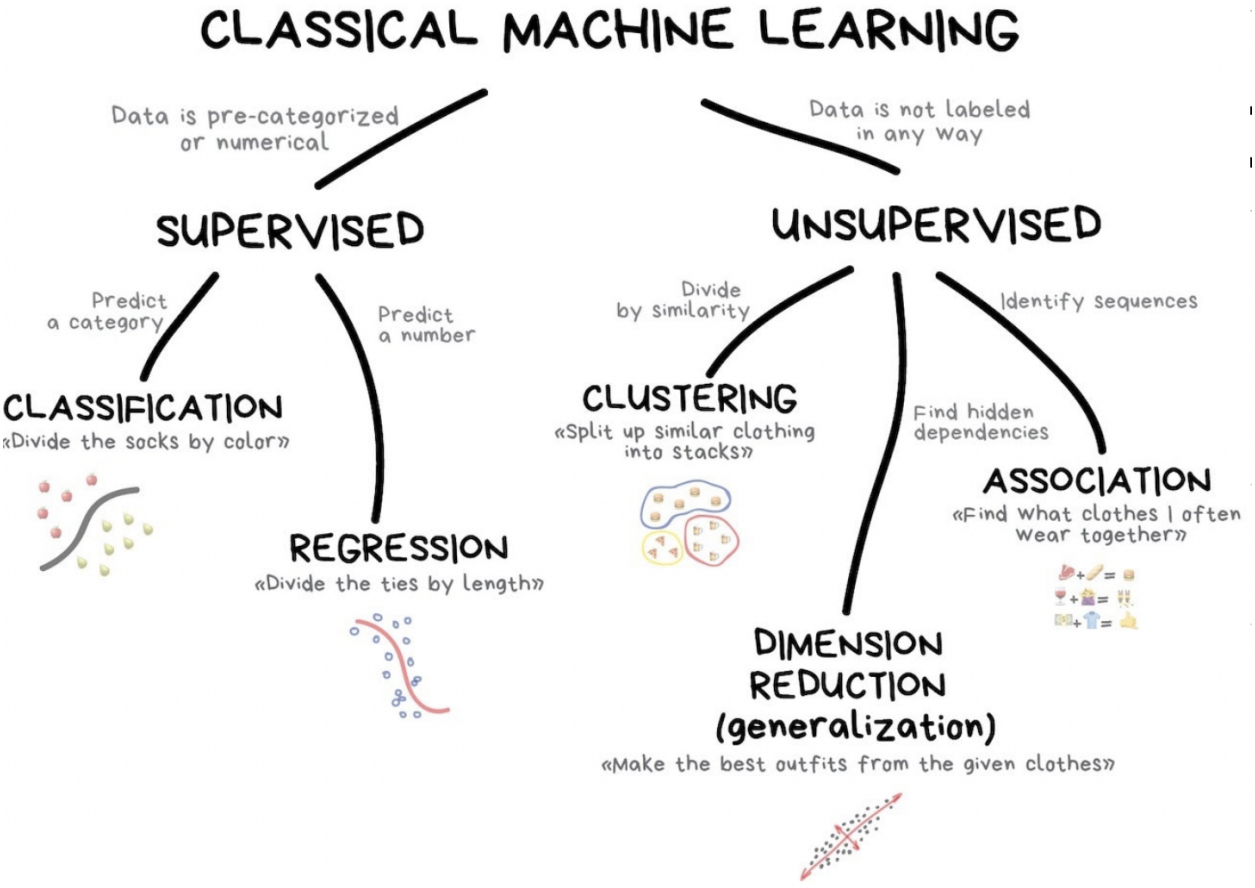
Introduccion al aprendizaje supervisado

Introducción a los Métodos Estadísticos

2021

Introucción al aprendizaje automatico



Aprendizaje supervisado

El objetivo es poder predecir/estimar los valores de una variable y en función de otras variables x.

Podemos pensar el problema de la siguiente manera

$$y \sim x_1 + x_2 + ... + x_J$$

donde

- y es la variable de salida/respuesta/dependiente
- x son las variables de entrada/predictoras/independientes/covariables/features

En el aprendizaje supervisado el objetivo es encontrar la mejor función f que prediga los valores de la variable y en función de las variables x basados en los datos.

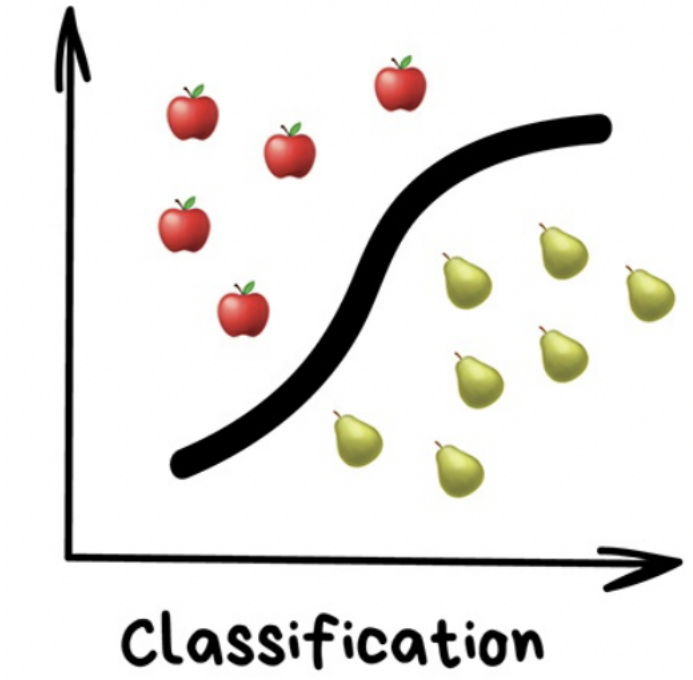
En otras palabras, $y = f(x) + e$ en donde f es una función que relaciona las variables de entrada x con la variable de salida y. Obviamente, la función f es desconocida y e es el término de error (lo que no puede ser explicado por f). La idea, es utilizando los datos poder estimar la función f para poder predecir los datos de la variable de salida, es decir, $\hat{y} = \hat{f}(x)$.

Es importante tener en cuenta que vamos a estimar la función f con los datos de entrenamiento y luego vamos a evaluar que tan bueno es nuestro algoritmo evaluando los datos de la x de los datos de test (los que nunca vio) en nuestra función estimada.

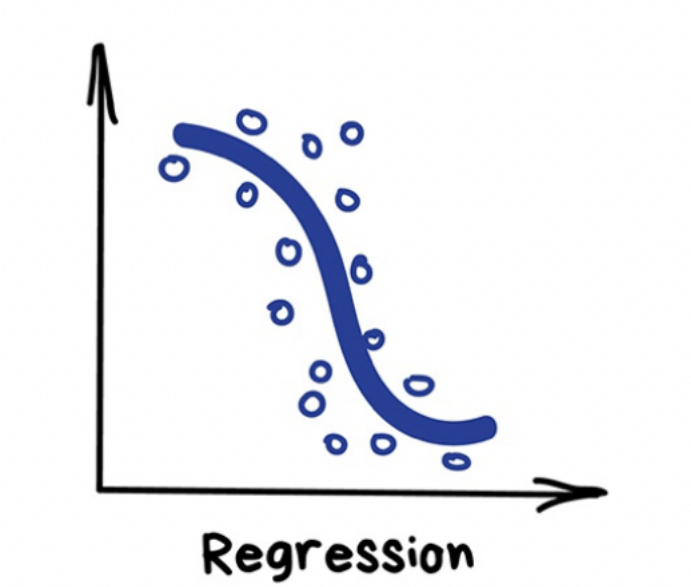
Como podemos ver el aprendizaje automático? Como un conjunto de métodos (modelos/algoritmos) para poder estimar f !!

Clasificación o regresión?

Los problemas de **clasificación** son aquellos en donde la variable de salida y es categórica. Por ejemplo, en el caso binario, el 1 es "tiene/es" y el 0 es "no tiene/no es". Por lo tanto, en los problemas de clasificación la idea es estimar la probabilidad de que $y = 1$ en función de las variables de entrada x, es decir, $P(y=1|x). Obviamente, esto, se puede extender a problemas en donde la variable de salida presenta muchas categorías.



La **regresión** es utilizada cuando la variable de salida y (la que queremos predecir/estimar) es numérica.



Dataset de entrenamiento y test

Lo primero que debemos hacer es separar nuestro dataset original en dos, llamados, "dataset de entrenamiento" y "dataset de test". El dataset de entrenamiento (train) consiste generalmente en una proporción grande del dataset original (e.g. 80%). En el dataset de entrenamiento vamos a utilizar a nuestro algoritmo para que pueda aprender de los datos. Finalmente, el dataset de test es utilizado para ver que tan bueno es nuestro algoritmo para predecir datos que nunca ha visto.

Perfomance de los algortimos

Para evaluar que tan bueno es nuestro algoritmo para predecir los valores de la y vamos a utilizar una serie de métricas, las cuales vamos a computarlas tanto para el dataset de entrenamiento como en el dataset de test.

Un modelo o algoritmo que tiene una perfomance pobre en el dataset de entrenamiento vamos a decir que tiene sub-ajuste (underfitting). Esto implica que el algoritmo no es capaz de generalizar la relación entre la variable de salida y y las variables de entrada x

Por otra parte, para un modelo o algoritmo que tenga una buena perfomance en los datos de entrenamiento pero un perfomance pobre en los datos de test, vamos a decir que tiene sobre-ajuste (overfitting). Esto se debe a que el modelo tiene un performance más bajo de lo esperado para datos nuevos (que nunca ha visto). Este problema es de los más comunes debido a que muchos algoritmos son capaces de "memorizar" los datos con los cuales lo estamos entrenando pero pierde la capacidad de generalizar.

Las métricas que vamos a utilizar van a depender del problema, es decir, si es un problema de clasificación o regresión. Para el caso de los problemas de clasificación vamos a utilizar la `accuracy` (proporción de observaciones que clasifica de forma correcta) y el error cuadrático medio o el R^2 para los problemas de regresión.

Boston Housing

Utilizamos el dataset `BostonHousing`. La descripción de las variable se presenta a continuación:

- `crim` per capita crime rate by town
- `zn` proportion of residential land zoned for lots over 25,000 sq. ft
- `indus` proportion of non-retail business acres per town
- `chas` Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- `nox` nitric oxide concentration (parts per 10 million)
- `rm` average number of rooms per dwelling
- `age` proportion of owner-occupied units built prior to 1940
- `dis` weighted distances to five Boston employment centers
- `rad` index of accessibility to radial highways
- `tax` full-value property tax rate per $10,000
- `ptratio` pupil-teacher ratio by town
- `b` proxi for proportion of African American by town
- `lstat` percentage of lower status of the population
- `medv` median value of owner-occupied homes in $1000s

El objetivo es predecir el valor de las casas, es decir, la variable de salida y es `medv` y el resto de las variables incluidas en el dataset son las variables de entrada x.

Para la predicción de la variable y vamos a utilizar tres algoritmos:

- modelos de regresión lineal (`lm`)
- arboles de regresión (`decision_tree`)
- bosques aleatorios (`randomforest`)

cargamos paquetes y el dataset

```
library(tidyverse)
library(tidymodels)
library(mlbench)
library(rpart.plot)
library(vip)
data(BostonHousing)
boston =BostonHousing
```

Si bien, los datos previos a realizar los algoritmos deben ser pre-procesados, en un principio, los ponemos "crudos", es decir, "como vienen". Es recomendable siempre hacer lo siguiente:

- eliminar variables explicativas o de entrada que se encuentren correlacionadas
- eliminar o imputar datos faltantes
- normalizar los datos

Separación o split del dataset

Separamos el dataset original de forma aleatoria en dos: uno de entrenamiento y otro de test. El dataset de entrenamiento es donde vamos a entrenar o estimar el algoritmo. Una vez realizado lo anterior, el perfomance del algoritmo, es decir, que tan bien predice para datos que NUNCA vio, lo vamos a hacer en los datos de test.

Definimos el dataset de test como un 20% de los datos originales

```
set.seed(1234)
boston_split = initial_split(boston,prop=0.80)
boston_split
```

```
## <Analysis/Assess/Total>
## <404/102/506>
```

Guardamos los dataset con nombres

```
# datos de entrenamiento
boston_train = training(boston_split)
# datos de test
boston_test = testing(boston_split)
```

Regresión lineal

Como primer paso hacemos la regresión lineal

```
boston_lm =linear_reg() %>%
          set_engine('lm') %>%
          fit(medv~., data=boston_train)
boston_lm %>% tidy()
```

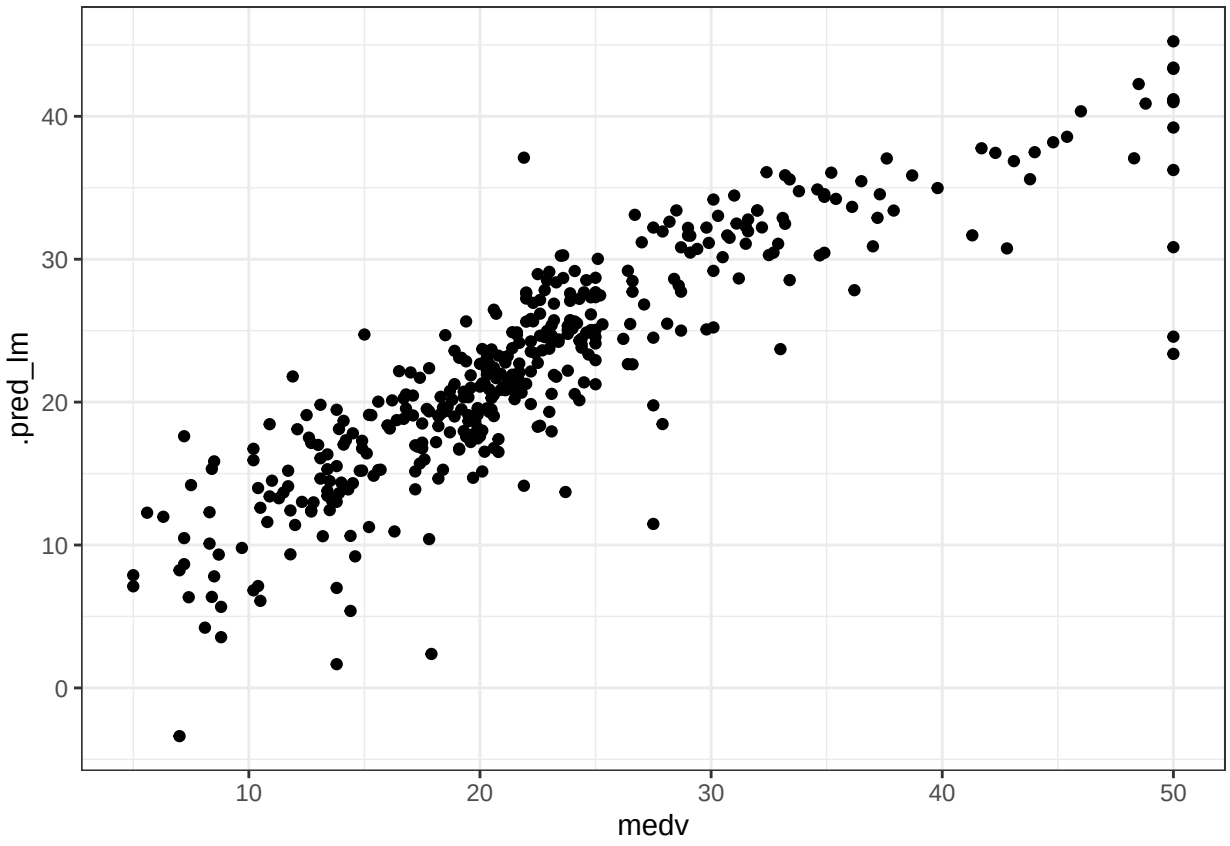
```
## # A tibble: 14 x 5
##    term         estimate std.error statistic  p.value
##    <chr>           <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)  32.4       5.71        5.67  2.74e- 8
##  2 crim         -0.111     0.0339     -3.26  1.20e- 3
##  3 zn            0.0456    0.0146      3.12  1.92e- 3
##  4 indus        -0.0218    0.0662     -0.328 7.43e- 1
##  5 chas1         2.88      0.925       3.11  2.00e- 3
##  6 nox         -17.1       4.11       -4.15  4.06e- 5
##  7 rm            4.26      0.470       9.07  5.77e-18
##  8 age          -0.0145    0.0147     -0.985 3.25e- 1
##  9 dis          -1.49      0.218      -6.82  3.39e-11
## 10 rad           0.276     0.0717      3.85  1.39e- 4
## 11 tax          -0.0124    0.00404    -3.08  2.24e- 3
## 12 ptratio      -0.893     0.145      -6.17  1.68e- 9
## 13 b             0.00921   0.00296     3.11  2.00e- 3
## 14 lstat        -0.405     0.0569     -7.12  5.25e-12
```

```
resultados_lm_train = boston_train %>% select(medv) %>%
  bind_cols(predict(boston_lm, boston_train)) %>%
  rename(.pred_lm=.pred)
resultados_lm_train  %>% head()
```

```
##     medv .pred_lm
## 284 50.0 45.24837
## 336 21.1 20.83368
## 406  5.0  7.88598
```

```
## 101 27.5 24.50937
## 492 13.6 13.30886
## 111 21.7 21.15149
```

```
resultados_lm_train %>% ggplot(aes(x=medv,y=.pred_lm))+geom_point()+theme_bw()
```



Computamos algunas métricas para evaluar que tan bueno es nuestro modelo

```
metrics(resultados_lm_train, truth = medv, estimate = .pred_lm)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       4.54
## 2 rsq     standard       0.759
## 3 mae     standard       3.17
```

Por defecto devuelve tres

$$\text{rmse} = \sqrt{\frac{1}{n} \times \sum (y_i - \hat{y}_i)^2}$$

$$\text{rsq} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

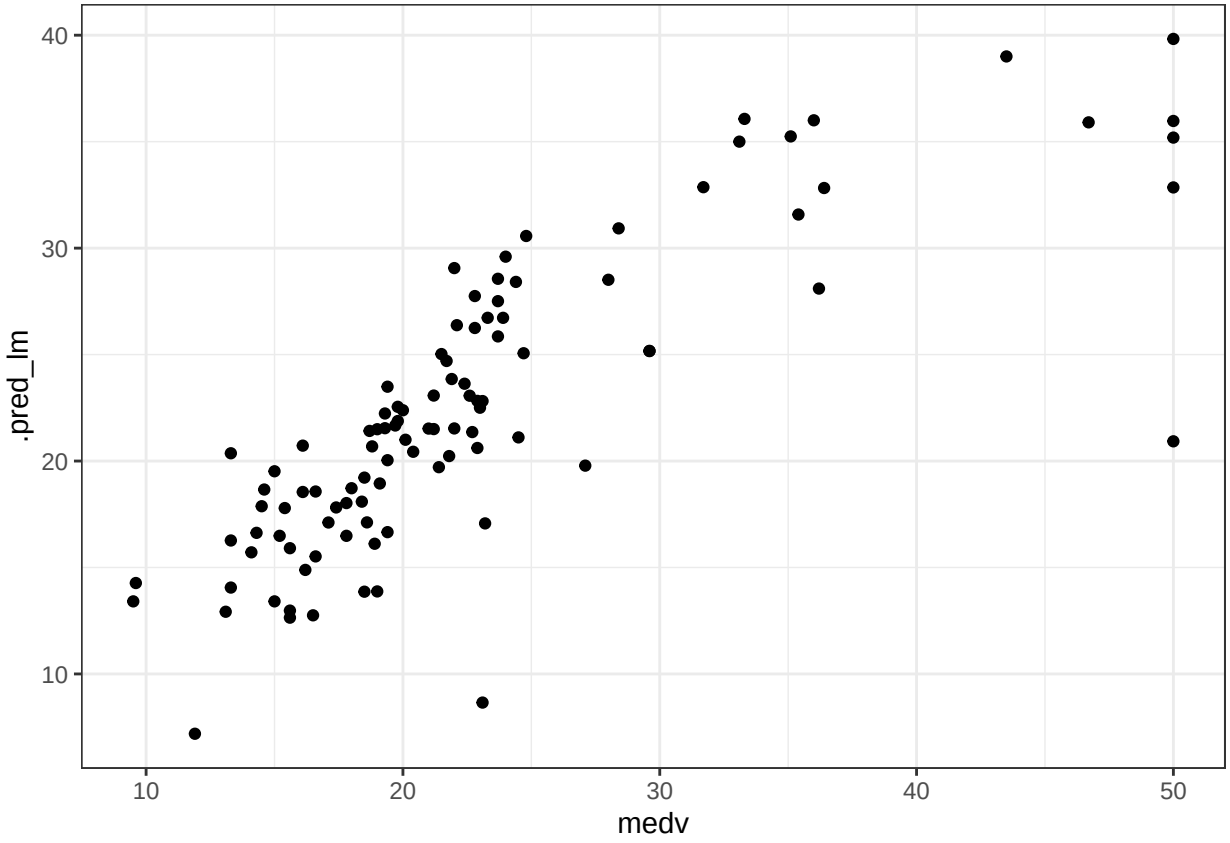
$$\text{mae} = \frac{1}{n} \times \sum |y_i - \hat{y}_i|$$

Evaluamos que tan bueno es nuestro modelo con datos que no utilizo para aprender o entrenar

```
resultados_lm_test= boston_test %>% select(medv) %>%
  bind_cols(predict(boston_lm, boston_test)) %>%
  rename(.pred_lm=.pred)
resultados_lm_test %>% head()
```

```
##    medv .pred_lm
## 1  24.0 29.59854
## 5  36.2 28.10004
## 7  22.9 22.82533
## 8  27.1 19.78399
## 9  16.5 12.75448
## 11 15.0 19.51924
```

```
resultados_lm_test %>% ggplot(aes(x=medv,y=.pred_lm))+geom_point()+theme_bw()
```




```
metrics(resultados_lm_test, truth = medv, estimate = .pred_lm)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
```

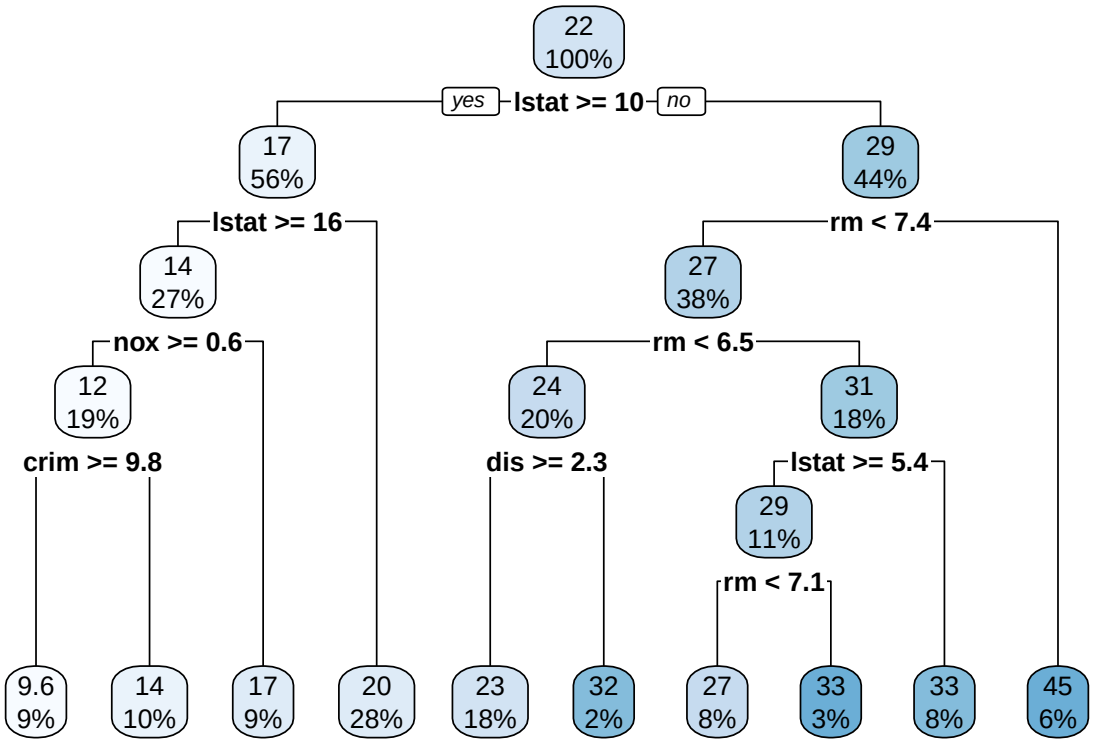
```
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       5.37
## 2 rsq     standard       0.647
## 3 mae     standard       3.46
```

Arboles de regresión

```
boston_tree = decision_tree() %>%
              set_engine('rpart') %>%
              set_mode('regression') %>%
              fit(medv~.,  data= boston_train)
```

Visualizamos el árbol que entrenamos

```
rpart.plot(boston_tree$fit, roundint = FALSE)
```

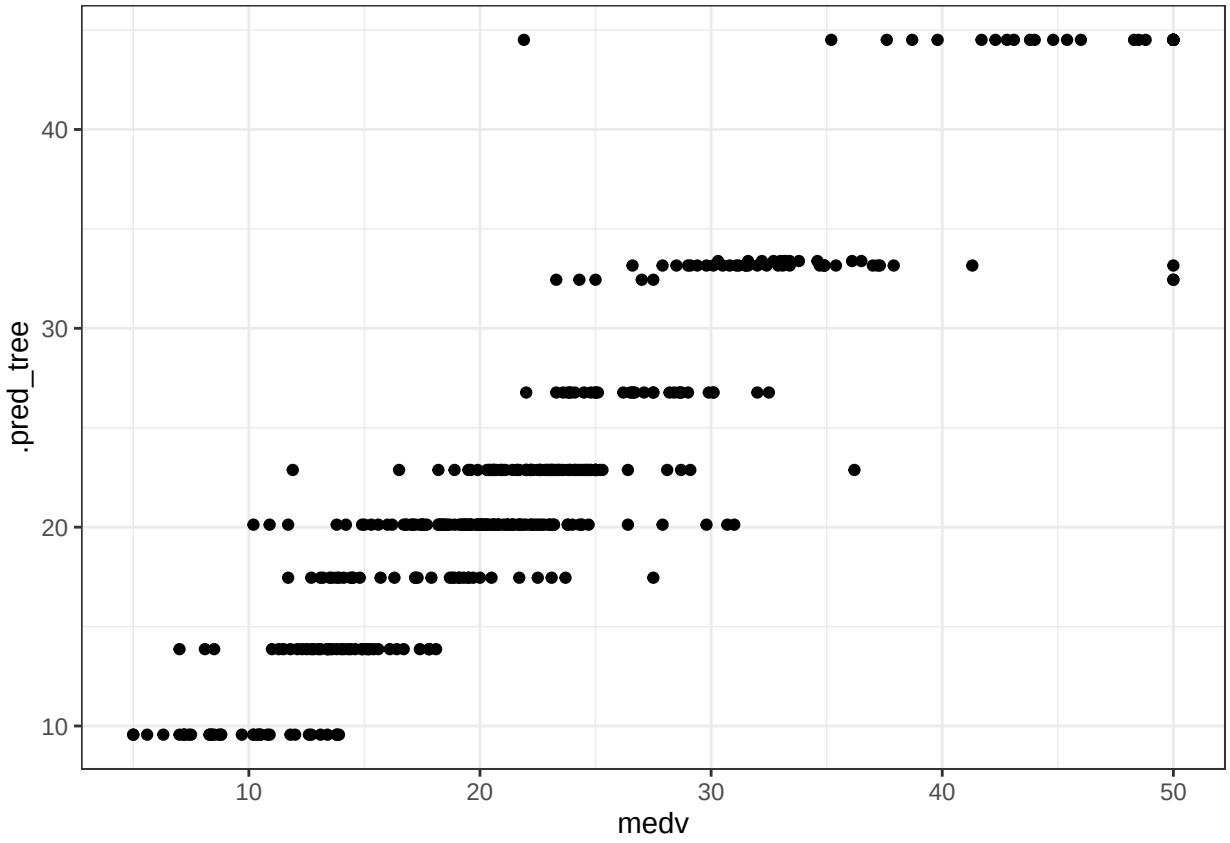


En este caso, la predicción $\hat{y}_i$ es el promedio de la variable y del nodo.

```
resultados_tree_train= boston_train %>% select(medv) %>%
  bind_cols(predict(boston_tree, boston_train)) %>%
  rename(.pred_tree=.pred)
resultados_tree_train %>% head()
```

```
##     medv .pred_tree
## 284 50.0  44.508000
## 336 21.1  22.876712
## 406  5.0   9.563889
## 101 27.5  26.770968
## 492 13.6  13.865000
## 111 21.7  20.124561
```

```
resultados_tree_train %>% ggplot(aes(x=medv,y=.pred_tree))+geom_point()+ theme_bw()
```



Evaluamos la performance del algoritmo en los datos de entrenamiento

```
metrics(resultados_tree_train, truth = medv, estimate = .pred_tree)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.73
## 2 rsq     standard       0.837
## 3 mae     standard       2.59
```

Evaluamos el algoritmo en los datos de test

```
resultados_tree_test= boston_test %>% select(medv) %>%
  bind_cols(predict(boston_tree, boston_test)) %>%
  rename(.pred_tree=.pred)
resultados_tree_test %>% head()
```

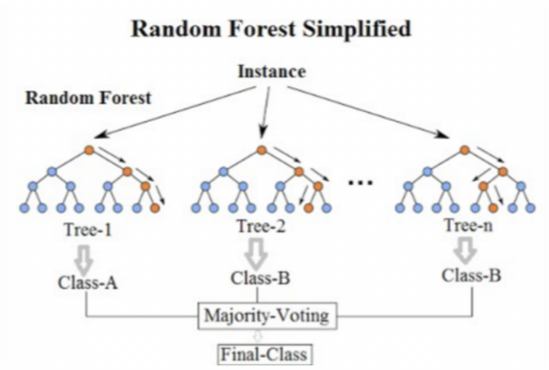
```
##    medv .pred_tree
## 1  24.0   33.15806
## 5  36.2   33.15806
## 7  22.9   20.12456
## 8  27.1   17.45714
## 9  16.5   17.45714
## 11 15.0   17.45714
```

```
metrics(resultados_tree_test, truth = medv, estimate = .pred_tree)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       4.37
## 2 rsq     standard       0.766
## 3 mae     standard       3.18
```

Random Forest

Random Forest o bosques aleatorios es un algoritmo que busca mejorar la perfomance en comparación a un único árbol. La idea? crear un bosque (muchos arboles) y luego predecir promediando los resultados de todos los arboles.



Por ultimo, aplicamos el algoritmo random forest. Hacemos un bosque con 1000 arboles o estimadores.

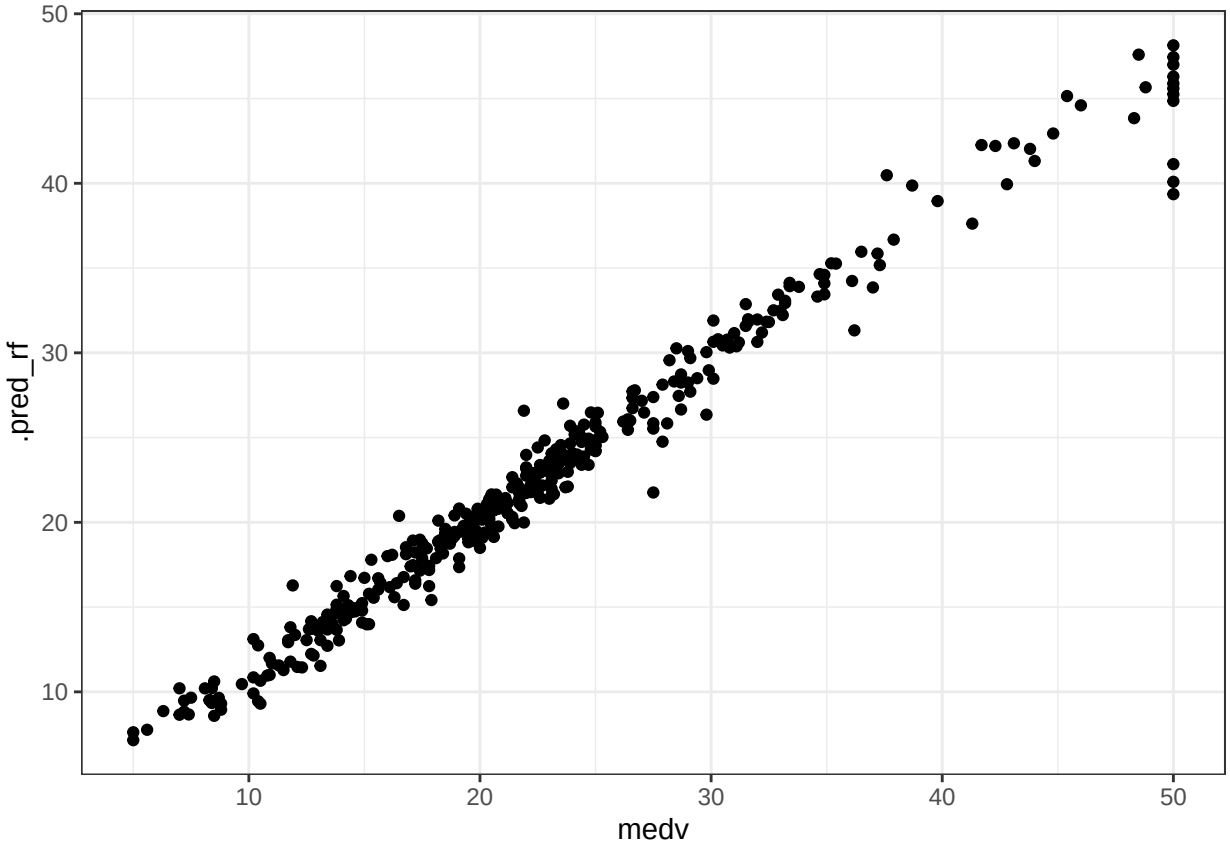
```
set.seed(1234)

boston_rf = rand_forest(trees = 1000) %>%
  set_engine('ranger',importance='impurity') %>%
  set_mode('regression') %>%
  fit(medv~., data=boston_train)
boston_rf
```

```
## parsnip model object
##
## Fit time:  236ms
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.trees = ~1000,      importance = ~"impurity", nu
##
## Type:                             Regression
## Number of trees:                  1000
## Sample size:                      404
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       11.27084
## R squared (OOB):                  0.8683085
```

```
resultados_rf_train= boston_train %>% select(medv) %>%
  bind_cols(predict(boston_rf, boston_train)) %>%
  rename(.pred_rf=.pred)
resultados_rf_train %>% head()
```

```
##     medv  .pred_rf
## 284 50.0 46.996990
## 336 21.1 20.798950
## 406  5.0  7.616569
## 101 27.5 25.845131
## 492 13.6 13.883002
## 111 21.7 21.390268
```

```
resultados_rf_train %>% ggplot(aes(x=medv,y=.pred_rf))+geom_point()+ theme_bw()
```




```
metrics(resultados_rf_train, truth = medv, estimate = .pred_rf)
```

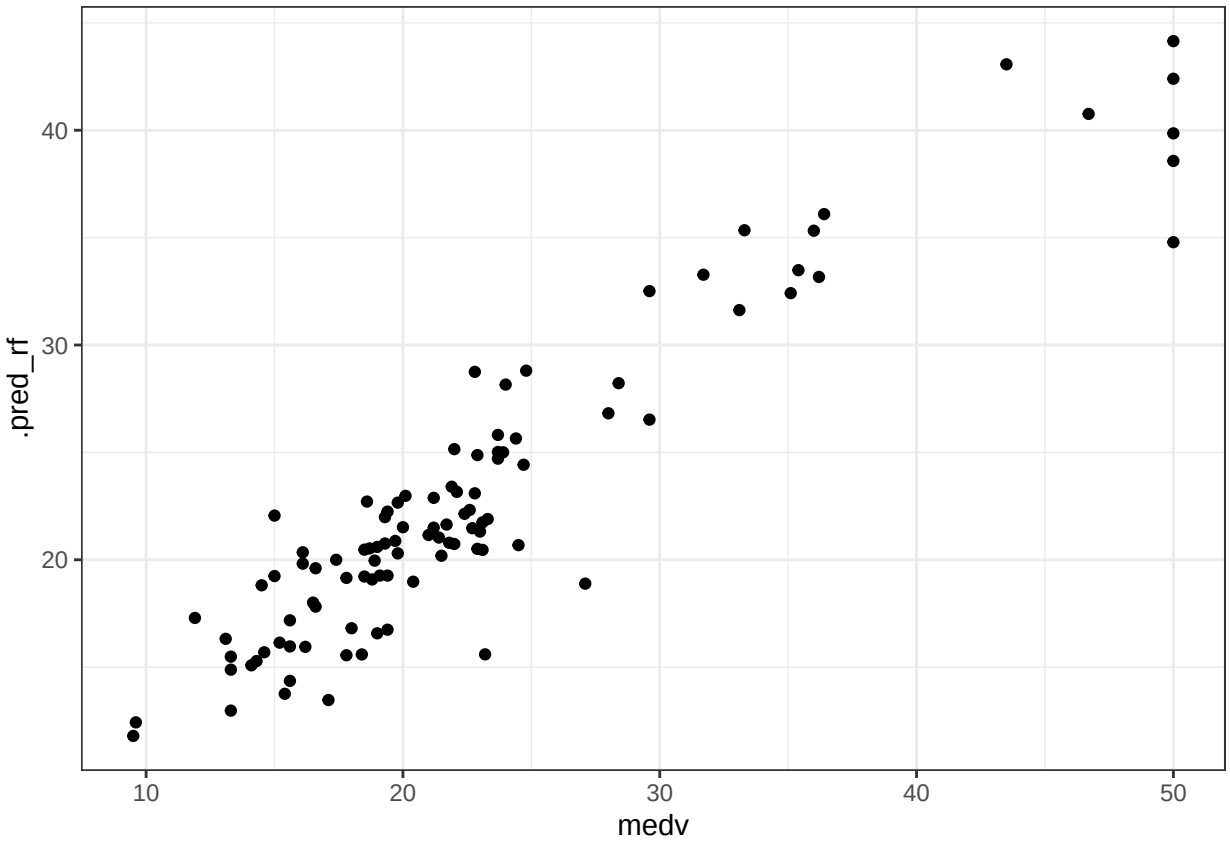
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       1.58
## 2 rsq     standard       0.977
## 3 mae     standard       1.02
```

Evaluamos el algoritmo en los datos de test

```
resultados_rf_test= boston_test %>% select(medv) %>%
  bind_cols(predict(boston_rf, boston_test)) %>%
  rename(.pred_rf=.pred)
resultados_rf_test %>% head()
```

```
##    medv .pred_rf
## 1  24.0 28.15739
## 5  36.2 33.16847
## 7  22.9 20.50869
## 8  27.1 18.88867
## 9  16.5 18.00676
## 11 15.0 19.24345
```

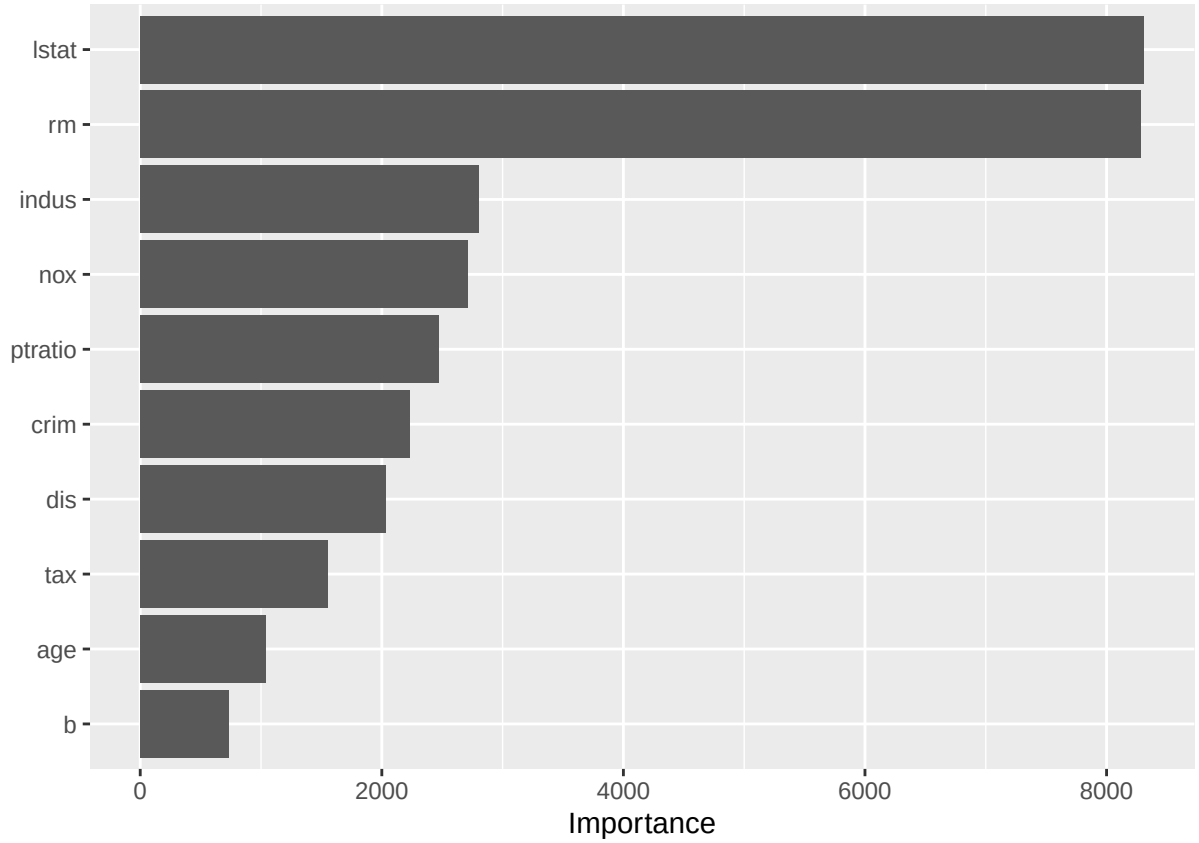
```
resultados_rf_test %>% ggplot(aes(x=medv,y=.pred_rf))+geom_point()+ theme_bw()
```




```
metrics(resultados_rf_test, truth = medv, estimate = .pred_rf)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.47
## 2 rsq     standard       0.870
## 3 mae     standard       2.43
```

```
boston_rf %>% vip()
```



Iris

La idea es poder **clasificar** la especie en base a algunas características de las flores

```
iris %>% head()
```

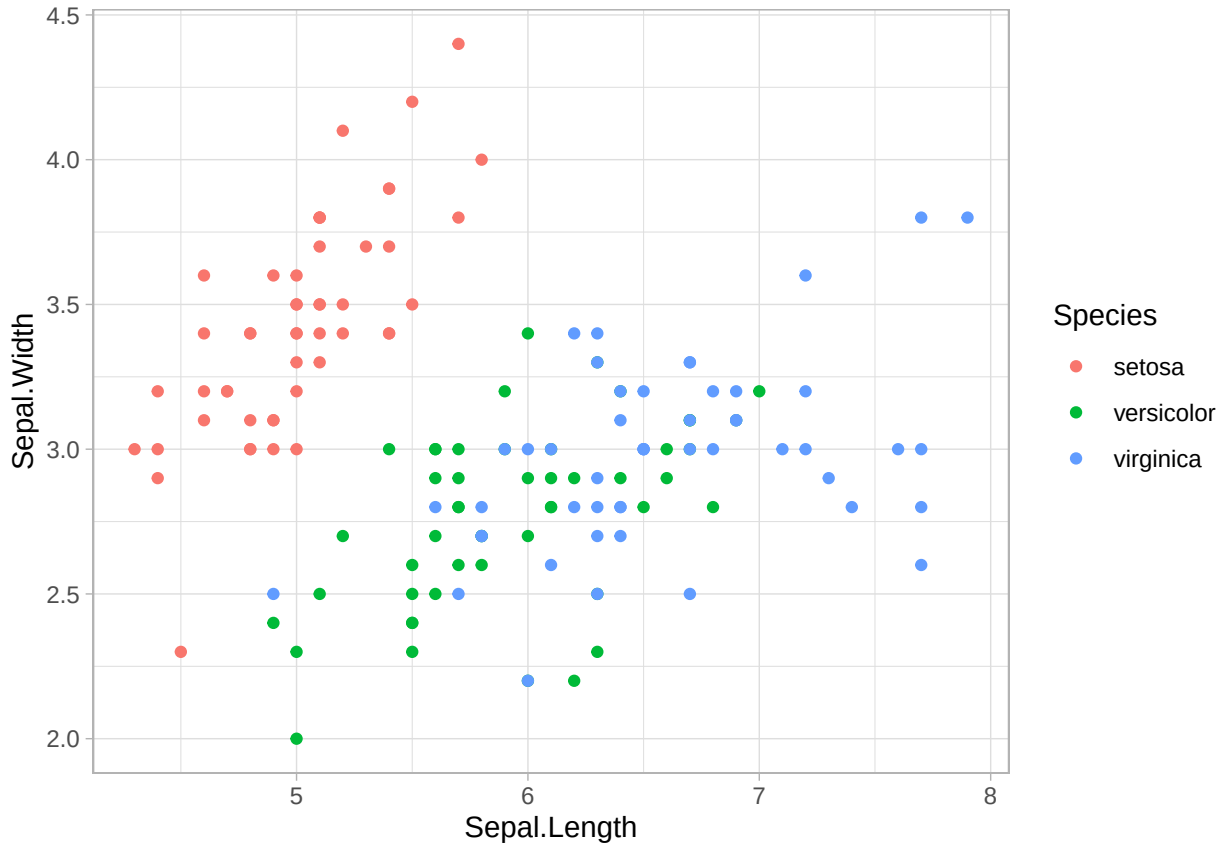
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5  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

- `sepal` length in cm
- `sepal` width in cm
- `petal` length in cm
- `petal` width in cm
- `Species` Setosa,versicolor, virginica

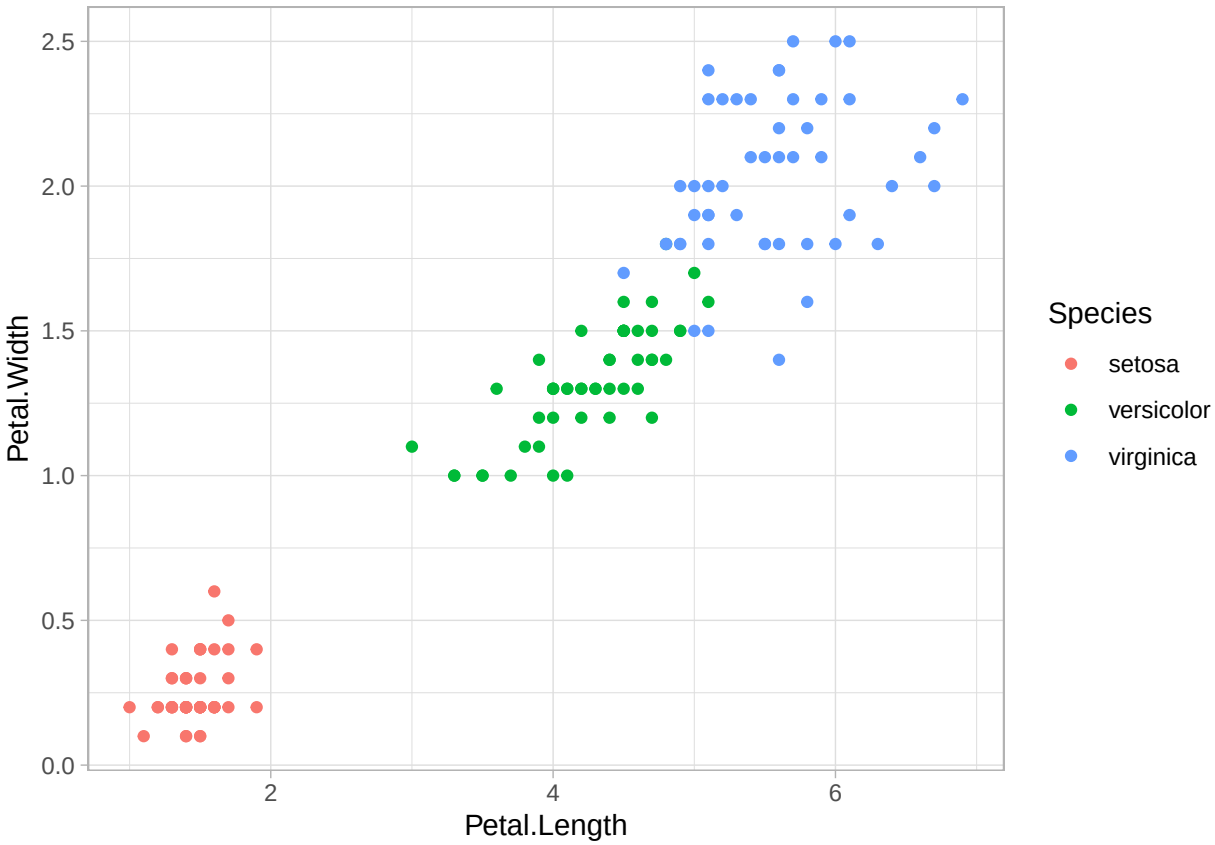
En este caso, la variable de salida y es la especie (`Species`) y el resto de las variables del dataset son las de entrada/explicativas x

Como paso previo, hacemos algunas visualizaciones para explorar un poco los datos

```
iris %>% ggplot(aes(x=Sepal.Length,
                    y=Sepal.Width,
                    color=Species))+geom_point() + theme_light()
```




```
iris %>% ggplot(aes(x=Petal.Length,
                    y=Petal.Width,
                    color=Species))+geom_point() + theme_light()
```



Separación o split del dataset

```
set.seed(12345)
iris_split = initial_split(data= iris, prop=0.8)
iris_split
```

```
## <Analysis/Assess/Total>
## <120/30/150>
```

```
# datos de entrenamiento
iris_train = training(iris_split)
# datos de test
iris_test = testing(iris_split)
```

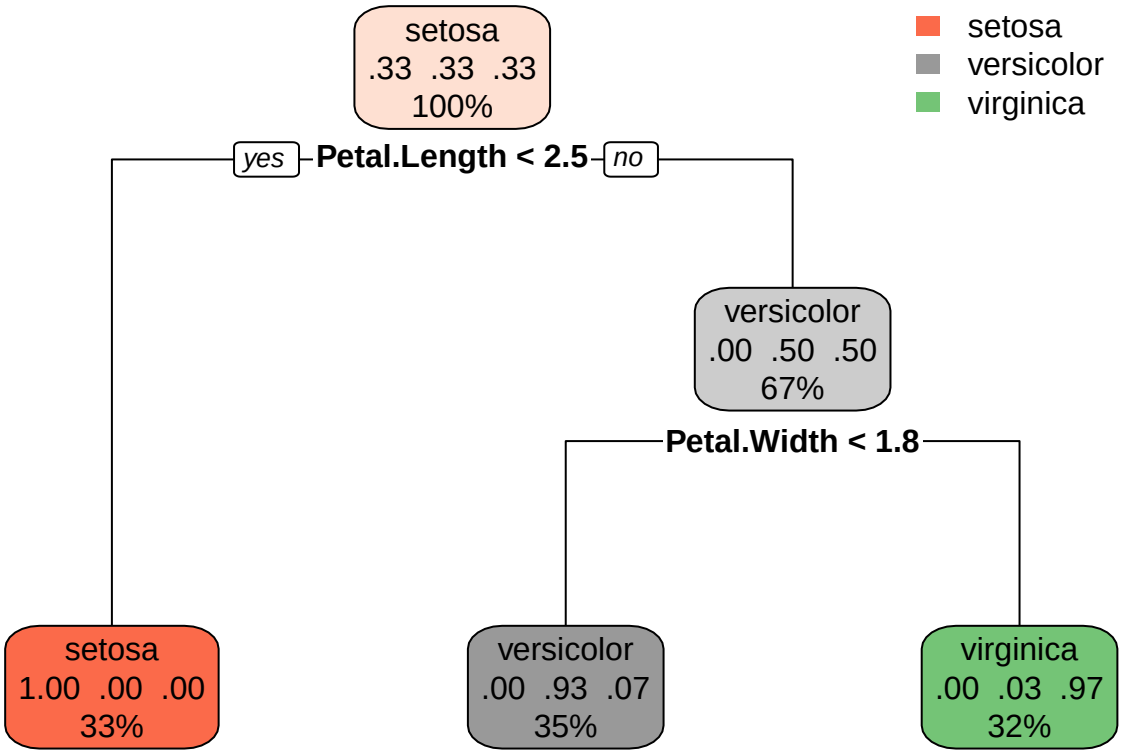
Arboles de clasificación.

En este caso, la predicción de la clase o categoría cuando utilizamos arboles es la moda (la clase con mayor frecuencia) dentro del nodo.

```
iris_tree = decision_tree() %>%
              set_engine('rpart') %>%
              set_mode('classification') %>%
              fit(Species~.,  data= iris_train)
```

Visualizamos el árbol que entrenamos

```
rpart.plot(iris_tree$fit, roundint = FALSE)
```




```
resultados_tree_train= iris_train %>% select(Species) %>%
  bind_cols(predict(iris_tree, iris_train)) %>%
  rename(.pred_tree=.pred_class)
resultados_tree_train %>% head()
```

```
##      Species .pred_tree
## 1  virginica  virginica
## 2 versicolor versicolor
## 3 versicolor versicolor
## 4 versicolor versicolor
## 5 versicolor versicolor
## 6 versicolor versicolor
```

Calculamos las métricas para los datos de entrenamiento

```
metrics(resultados_tree_train, truth = Species, estimate = .pred_tree)
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.967
## 2 kap      multiclass     0.95
```

La métrica `accuracy` nos dice el porcentaje de observaciones que fueron clasificadas de forma correcta.

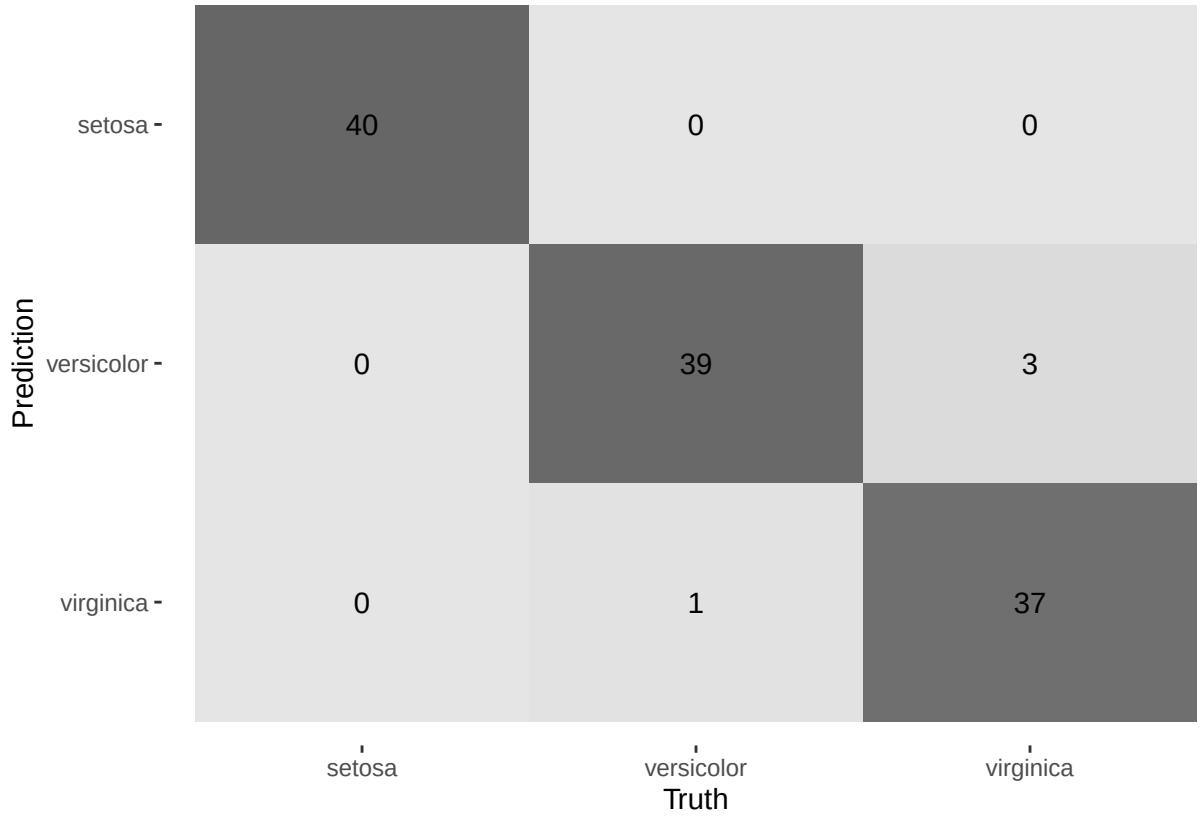
Hacemos la **matriz de confusión**

```
cm = conf_mat(resultados_tree_train, truth = Species, estimate = .pred_tree)
cm
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         40          0         0
##   versicolor      0         39         3
##   virginica       0          1        37
```

Visualizamos la matriz de confusión

```
autoplot(cm, type = "heatmap")
```



Evaluamos el algoritmo en los datos de test

```
resultados_tree_test = iris_test %>% select(Species) %>%
  bind_cols(predict(iris_tree, iris_test)) %>%
  rename(.pred_tree=.pred_class)
resultados_tree_test%>% head()
```

```
##   Species .pred_tree
```

```
## 1  setosa     setosa
## 2  setosa     setosa
## 3  setosa     setosa
## 4  setosa     setosa
## 5  setosa     setosa
## 6  setosa     setosa
```

Calculamos las métricas para los datos de test

```
metrics(resultados_tree_test, truth = Species, estimate = .pred_tree)
```

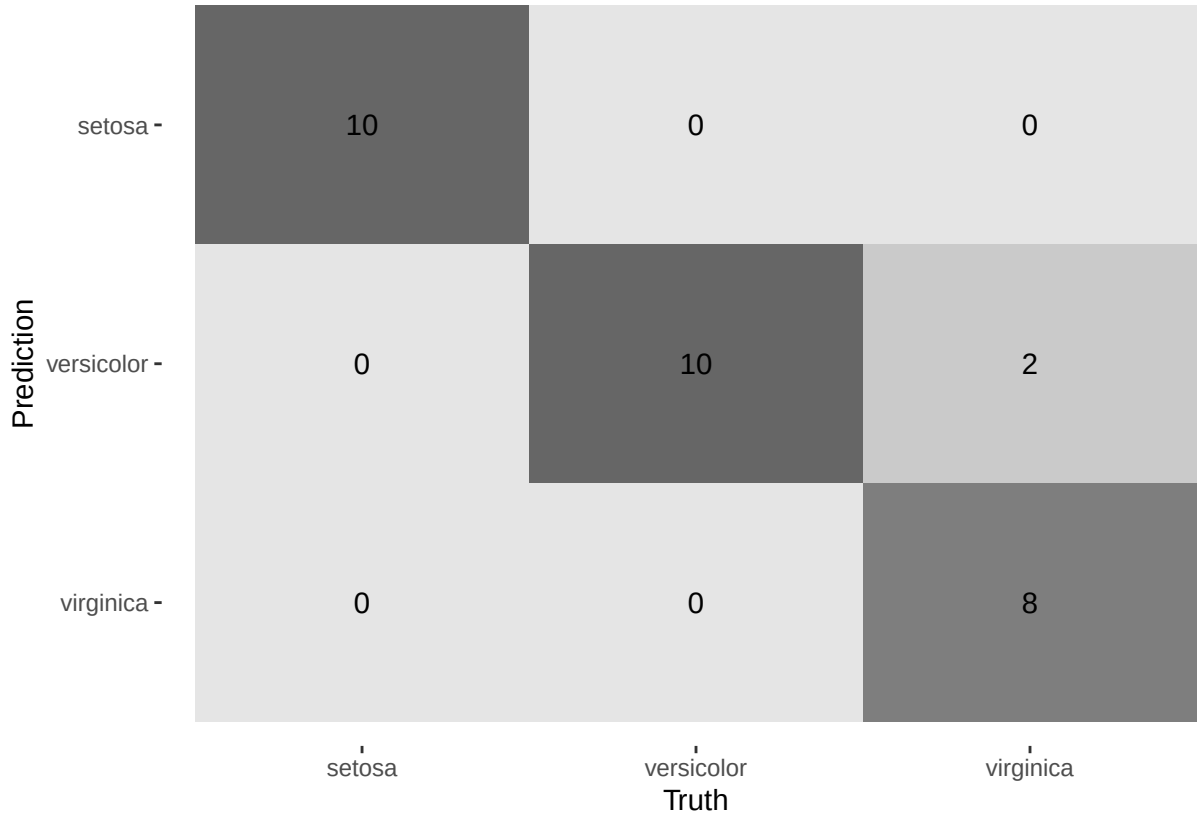
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.933
## 2 kap      multiclass     0.9
```

```
cm_test = conf_mat(resultados_tree_test, truth = Species, estimate = .pred_tree)
cm_test
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         2
##   virginica       0          0         8
```

Visualizamos la matriz de confusión

```
autoplot(cm_test, type = "heatmap")
```



Random Forest

En este caso, la predicción de la clase se realiza por el voto mayoritario.

```
set.seed(1234)
iris_rf = rand_forest(trees = 1000) %>%
  set_engine('ranger') %>%
  set_mode('classification') %>%
  fit(Species~., data=iris_train)
iris_rf
```

```
## parsnip model object
##
## Fit time:  17ms
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.trees = ~1000,      num.threads = 1, verbose = F
##
## Type:                             Probability estimation
## Number of trees:                  1000
## Sample size:                      120
## Number of independent variables:  4
## Mtry:                             2
```

```
## Target node size:                 10
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.02828274
```

```
resultados_rf_train = iris_train %>% select(Species) %>%
  bind_cols(predict(iris_rf, iris_train)) %>%
  rename(.pred_tree=.pred_class)
resultados_rf_train%>% head()
```

```
##      Species .pred_tree
## 1  virginica  virginica
## 2 versicolor versicolor
## 3 versicolor versicolor
## 4 versicolor versicolor
## 5 versicolor versicolor
## 6 versicolor versicolor
```

Calculamos las métricas para los datos de entrenamiento

```
metrics(resultados_rf_train, truth = Species, estimate = .pred_tree)
```

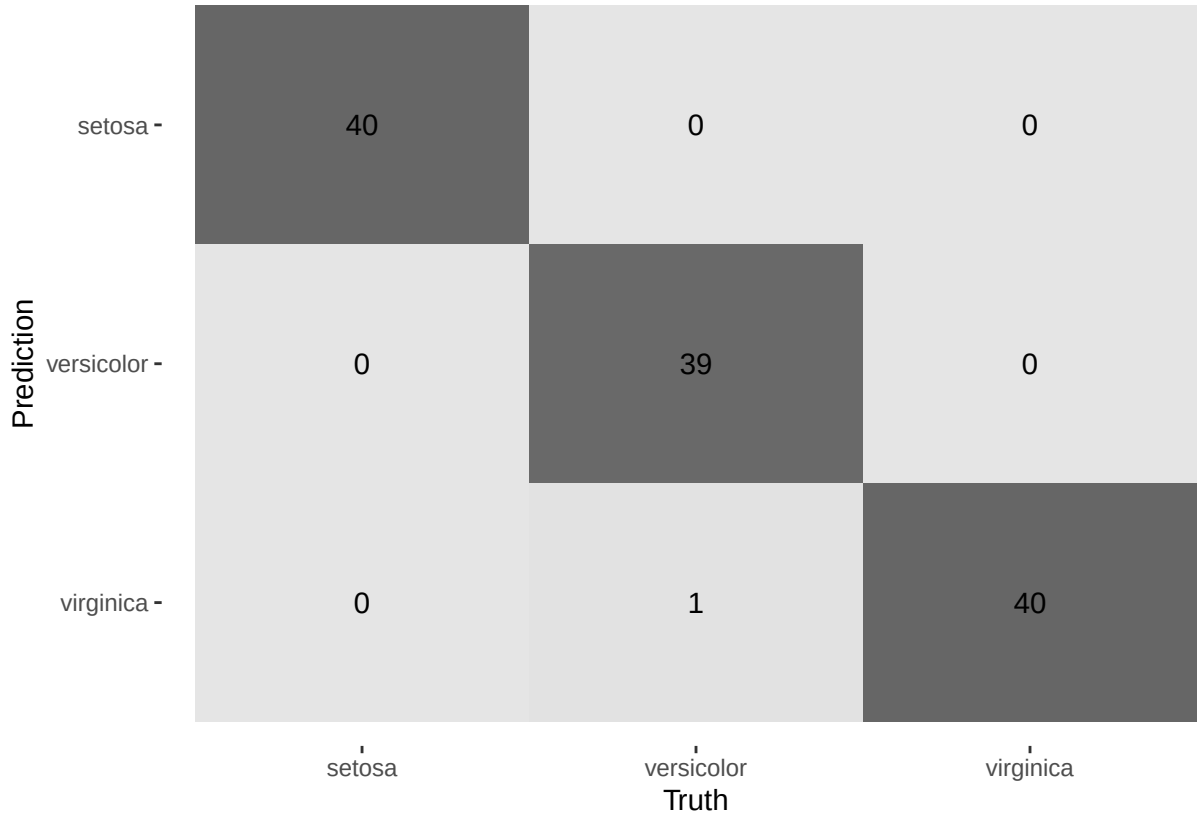
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.992
## 2 kap      multiclass     0.988
```

```
cm_train = conf_mat(resultados_rf_train, truth = Species, estimate = .pred_tree)
cm_train
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         40          0         0
##   versicolor      0         39         0
##   virginica       0          1        40
```

Visualizamos la matriz de confusión

```
autoplot(cm_train, type = "heatmap")
```



Ídem, pero para los datos del test

```
resultados_rf_test = iris_test %>% select(Species) %>%
  bind_cols(predict(iris_rf, iris_test)) %>%
  rename(.pred_tree=.pred_class)
resultados_rf_test %>% head()
```

```
##   Species .pred_tree
## 1  setosa     setosa
## 2  setosa     setosa
## 3  setosa     setosa
## 4  setosa     setosa
## 5  setosa     setosa
## 6  setosa     setosa
```

Calculamos las métricas para los datos de entrenamiento

```
metrics(resultados_rf_test, truth = Species, estimate = .pred_tree)
```

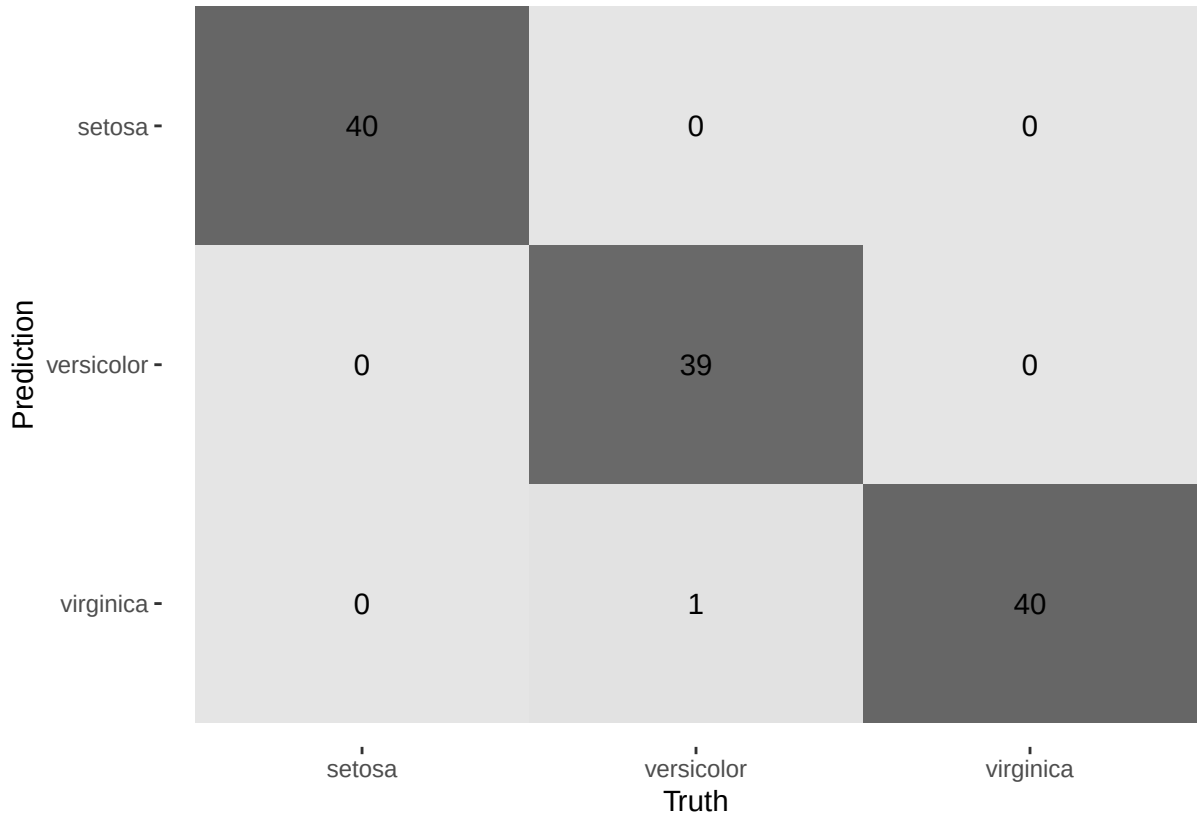
```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.933
## 2 kap      multiclass     0.9
```

```
cm_test = conf_mat(resultados_rf_test, truth = Species, estimate = .pred_tree)
cm_test
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         2
##   virginica       0          0         8
```

Visualizamos la matriz de confusión

```
autoplot(cm_train, type = "heatmap")
```



Haciendo recetas.

Antes de realizar cualquier algoritmo, es necesario hacer un pre-procesamiento a los datos. La versión corta es:

- definir la variable de salida y y las variables predictoras x
- remover o imputar datos faltantes (si es que los hay!)
- escalar los datos (si es necesario!)
- convertir las variables de tipo factor en indicadoras (si es que hay y si es que es necesario)