Elegimos trabajar con el dataset **"ChickWeight"**, que registra el crecimiento de pollos a lo largo del tiempo según la dieta que recibieron. Es un conjunto de datos muy utilizado en estadística y machine learning porque combina variables **numéricas** (como el peso y el tiempo) con variables **categóricas** (como la dieta), y presenta una evolución temporal que lo hace interesante para modelar.

En nuestro caso, nos enfocamos en la variable **"Time"** como variable dependiente. Esta indica la cantidad de días transcurridos desde el inicio del experimento. El objetivo de predecir el tiempo está relacionado con entender **cuándo** se alcanzan ciertos pesos, o cómo influye la dieta en la velocidad de crecimiento de los pollos. Es un problema de **regresión**, donde buscamos estimar el tiempo necesario para alcanzar un determinado peso, bajo distintas condiciones.

Esta variable es clave para analizar la relación entre el **peso** y el **tiempo**, y cómo esa relación varía según la dieta aplicada. A partir de este punto, fuimos aplicando distintos modelos y técnicas de análisis para entender mejor los datos y evaluar el rendimiento de distintos algoritmos de predicción y clasificación.

Lo primero que hicimos fue cargar las librerías correspondientes y luego comenzamos con la preparación de los datos y la partición de train y test.

PREPARACIÓN DE LOS DATOS

Cargamos el dataset **"ChickWeight"** y luego mostramos un resumen estadístico básico de las variables:

- Mínimo, máximo, media y cuartiles para weight y Time.
- Conteo por categoría para la variable Diet.
- Y te da una idea de la distribución de Chick (pollitos individuales).

Luego, mostramos las primeras 5 filas del dataset, lo que sirve para ver cómo están estructurados los datos (columnas como weight, Time, Chick, y Diet).

```
#set de datos "ChickWeight"
data(ChickWeight)
summary(ChickWeight)
head(ChickWeight,5)
```

```
     weight           Time           Chick     Diet
 Min.   : 35.0   Min.   : 0.00   13     : 12   1:220
 1st Qu.: 63.0   1st Qu.: 4.00   9      : 12   2:120
 Median :103.0   Median :10.00   20     : 12   3:120
 Mean   :121.8   Mean   :10.72   10     : 12   4:118
 3rd Qu.:163.8   3rd Qu.:16.00   17     : 12
 Max.   :373.0   Max.   :21.00   19     : 12
                                 (Other):506
```

A nfnGroupedData: 5 × 4

	weight <dbl>	Time <dbl>	Chick <ord>	Diet <fct>
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1

Con este código obtuvimos el tamaño total de la muestra, es decir, cuántas observaciones hay en el dataset ChickWeight. En este caso, el resultado fue 578, lo que significa que tenemos 578 registros, donde cada uno representa el peso de un pollo en un momento determinado del experimento.

```
len_muestra<-nrow(ChickWeight) #tamaño de la muestra
len_muestra
```

Luego, definimos una muestra aleatoria para dividir el dataset en un 80% para entrenamiento y un 20% para prueba. Usamos **set.seed(13)** para asegurar que la aleatoriedad sea reproducible, es decir, que si alguien vuelve a correr el código, obtenga la misma partición. Luego generamos una **p**ermutación aleatoria de los índices del dataset con la función **sample(1:len_muestra)**. Esto nos permitió barajar los datos antes de separarlos, evitando sesgos en la selección.

Calculamos el 80% del total para definir la cantidad de datos de entrenamiento, que nos dio un total de **462 observaciones**, y usamos esos primeros índices para construir el conjunto de entrenamiento. Los 116 registros restantes (20%) fueron asignados al conjunto de prueba, que se utilizará para evaluar la performance de cada modelo.

Esta división nos permite entrenar los algoritmos con una parte representativa de los datos y luego comprobar qué tan bien generalizan al enfrentarse a datos nuevos.

```
set.seed(13)
#Definimos los sets train-test (en este caso será 80%-20%)

sample_muestra <- sample(1:len_muestra) #muestra aleatoria
print(paste("Muestra aleatoria:",len_muestra))
sample_muestra

len_train <- floor(len_muestra * 0.8) #Largo dataset train - Aquí se
define la proporción train/test
print(paste("Longitud muestra train:", len_train))
sample_train <- sample_muestra[1:len_train] #muestra train
sample_train

sample_test <- sample_muestra[(len_train + 1):len_muestra] #muestra
test
len_test<-length(sample_test) #Largo dataset test
print(paste("Longitud muestra test:", len_test))
sample_test
```

Una vez divididos los datos, creamos los conjuntos de entrenamiento y prueba usando los índices previamente generados. Para eso seleccionamos las filas correspondientes del dataset original.

Luego, definimos qué variables iban a ser utilizadas como predictoras y cuál como objetivo. En este caso, elegimos usar weight y Time como variables predictoras, y Diet como variable a predecir, es decir, la clase que queremos que el modelo aprenda a identificar.

Además, escalamos las variables predictoras con la función scale() para que tengan la misma escala. Finalmente, convertimos la variable Diet en un factor categórico para que los algoritmos de clasificación la puedan interpretar correctamente:

```
# Crear datasets de entrenamiento y prueba
ChickWeight_train <- ChickWeight[sample_train, ]
ChickWeight_test <- ChickWeight[sample_test, ]

# Seleccionar variables predictoras y objetivo
ChickWeight_train_dataset <- scale(ChickWeight_train[, c("weight",
"Time")])
ChickWeight_test_dataset <- scale(ChickWeight_test[, c("weight",
"Time")])
ChickWeight_train_output <- factor(ChickWeight_train$Diet)
ChickWeight_test_output_real <- factor(ChickWeight_test$Diet)
```

KNN

El primer modelo que aplicamos fue el algoritmo **k-Nearest Neighbors (k-NN)**, que es un clasificador supervisado basado en la distancia entre observaciones. En este caso usamos la librería class de R.

Establecimos el valor de **k = 5**, es decir, que el modelo va a clasificar cada caso nuevo en función de las 5 observaciones más cercanas en el conjunto de entrenamiento.

Luego generamos una matriz de confusión, que nos permitió comparar las predicciones del modelo con las clases reales del conjunto de prueba. A partir de esta matriz, calculamos el accuracy, es decir, el porcentaje de predicciones correctas que realizó el modelo: Esto nos dio una primera idea de qué tan bien se comporta el algoritmo k-NN en la clasificación de la dieta en base al peso y al tiempo.

El modelo obtuvo un accuracy de aproximadamente 40.5%, lo cual significa que predijo correctamente el 40.5% de los casos del conjunto de prueba.

```
[1] "Matriz de confusión:"
                           ChickWeight_test_output_real
ChickWeight_test_output_knn  1  2  3  4
                           1 33  7  6  1
                           2  7  5 11  6
                           3  4  2  1  8
                           4  5  7  5  8
[1] "Accuracy: 0.405172413793103"
```

```
# Ajustar el modelo k-NN
k <- 5  # Puedes ajustar el valor de k después
ChickWeight_test_output_knn <- knn(
                            train = ChickWeight_train_dataset,
                            cl = ChickWeight_train_output,
                            test = ChickWeight_test_dataset,
                            k = k
                            )

# Crear la matriz de confusión
mc_knn <- table(ChickWeight_test_output_knn,
ChickWeight_test_output_real)
print("Matriz de confusión:")
print(mc_knn)

# Calcular el accuracy del modelo
ac_knn <- mean(ChickWeight_test_output_knn ==
ChickWeight_test_output_real)
print(paste("Accuracy:", ac_knn))
```

Una parte clave al trabajar con el algoritmo **k-Nearest Neighbors (k-NN)** es elegir el número adecuado de vecinos (*k*) que se consideran para hacer una predicción. Un valor muy bajo puede hacer que el modelo sea muy sensible al ruido (sobreajuste), mientras que un valor demasiado alto puede hacerlo demasiado general (subajuste).

Para encontrar el valor óptimo de *k*, implementamos un experimento en el que se evaluó el rendimiento del modelo para valores de k desde 1 hasta 100. En cada iteración:

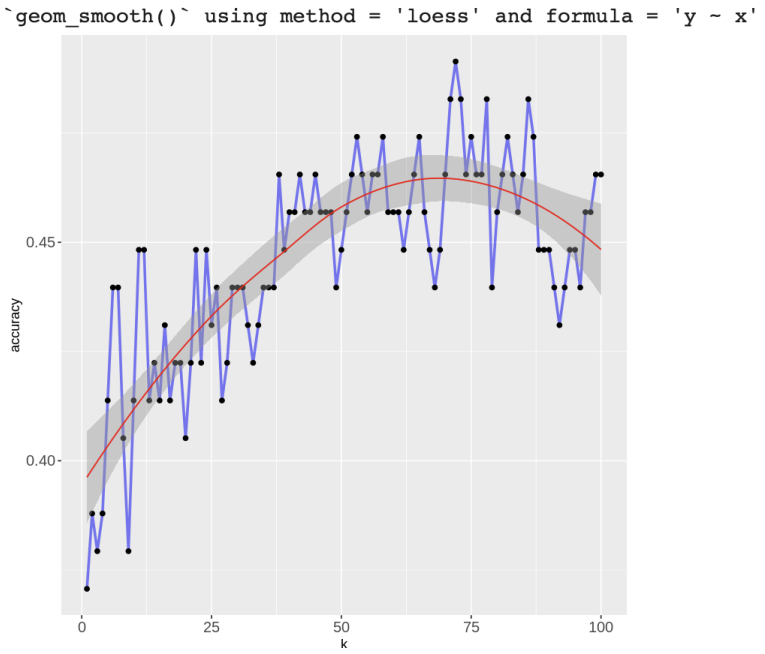
- Se entrenó el modelo k-NN con un valor específico de *k*.
- Se calculó la precisión (accuracy) del modelo en el conjunto de test.
- Se almacenaron los resultados en un data frame para analizarlos.

Luego, se realizó una visualización gráfica con ggplot2, en la que:

- El eje X representa los diferentes valores de *k* probados.
- El eje Y muestra el accuracy obtenido para cada uno.
- Se agregó una línea azul que conecta los puntos y una línea roja suavizada para observar la tendencia general del comportamiento del modelo.

Esta visualización nos permitió observar cómo varía el rendimiento del modelo con diferentes valores de *k* y facilita la identificación del valor de k que maximiza el rendimiento.

```r
#En este tipo de algoritmos, es común querer encontrar un valor de k vecinos óptimo. Para ello:

set.seed(13)

k <- 1:100

knn_kvalues <- data.frame(k, accuracy = 0)

for(n in k){

 ChickWeight_test_output_knn <- knn(

                            train = ChickWeight_train_dataset,

                            cl = ChickWeight_train_output,

                            test = ChickWeight_test_dataset,

                            k = n

                            )

knn_kvalues$accuracy[n] <- mean(ChickWeight_test_output_knn == ChickWeight_test_output_real)

}

#Resultados:

 ggplot(knn_kvalues) +

 aes(k, accuracy) +

 geom_line(colour="blue",lwd = 1, alpha = 0.5) +

 geom_point(colour="black") +

 geom_smooth(colour="red", lwd = 0.5) +

   theme(

         axis.text.x = element_text(size = 12),

         axis.text.y = element_text(size = 12)

         )
```



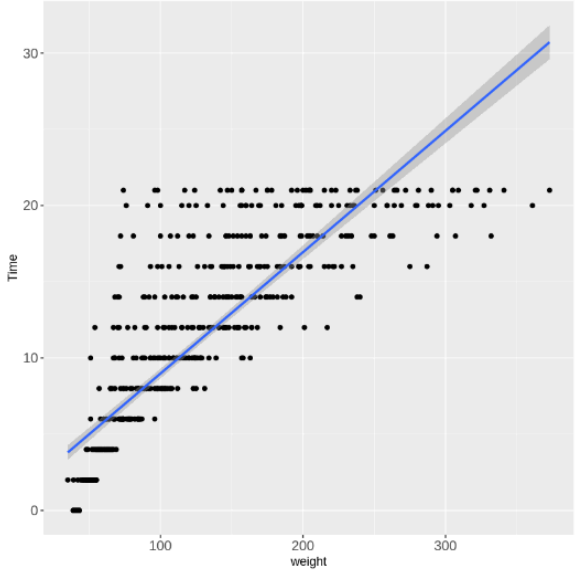
REGRESIÓN LINEAL

En este gráfico se representa la relación entre el peso de los pollitos (weight) y el tiempo (Time) en días.

Cada punto negro representa una observación individual en el conjunto de datos. La línea azul representa el ajuste de un modelo de regresión lineal simple, que permite visualizar la tendencia general.

Como se puede observar, existe una relación positiva entre el tiempo y el peso: a medida que pasan los días, el peso de los pollitos tiende a aumentar. Esto es consistente con lo esperado biológicamente, ya que los animales crecen con el tiempo.

Este gráfico es útil como análisis exploratorio inicial antes de aplicar modelos más complejos, ya que ayuda a visualizar si la relación entre variables es más o menos lineal y si hay patrones generales o puntos atípicos.



```r
#Partimos del gráfico visto en la introducción a R
ggplot(ChickWeight, aes(weight, Time)) +
 geom_point(colour="black") +
   theme(
        axis.text.x = element_text(size = 12),
        axis.text.y = element_text(size = 12)
        ) +

 geom_smooth(method = "lm")
```

En este caso, utilicé un modelo de regresión lineal múltiple para predecir el peso de los pollitos en función del tiempo (Time) y la identificación del pollito (Chick).

Si bien la regresión lineal es un modelo de tipo continuo, a partir de las predicciones realicé un proceso de discretización, dividiendo los valores predichos en tres rangos, con el fin de asignar una clase correspondiente a cada dieta.

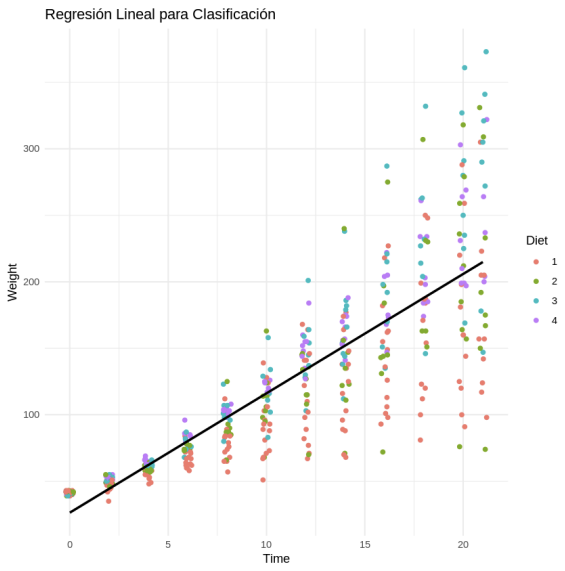
De esta forma, pude evaluar el modelo como si fuera un clasificador, utilizando una matriz de confusión y calculando el accuracy. Este método no es lo más convencional para clasificación, pero es válido para comparar modelos.

También generé un gráfico que muestra cómo se ajusta la regresión sobre los datos, con una línea que indica la tendencia general del crecimiento del peso en función del tiempo.

Aunque el modelo es de tipo regresivo (es decir, predice un valor numérico), discreticé las predicciones en tres intervalos utilizando cuantiles, y a cada uno le asigné una etiqueta correspondiente a una dieta. De esa manera, transformé la salida del modelo en una clasificación multiclase. Como se puede observar, el modelo nunca predijo correctamente la clase 4, lo cual es un indicador de desequilibrio o que los valores de peso no permitieron diferenciar adecuadamente esa clase al ser transformados.

El accuracy final del modelo fue de aproximadamente 28.6%, lo que representa un rendimiento por debajo del azar (que sería de 25% para 4 clases). Esto sugiere que, aunque la regresión lineal puede dar una idea general de tendencia, no es adecuada como clasificador directo en este caso, al menos no sin un procesamiento adicional o un enfoque diferente.

```
[1] "Matriz de Confusión:"
                          ChickWeight_test_output_rlineal
ChickWeight_test_output_real  1  2  3  4
                           1 18 16 12  0
                           2  6  7  8  0
                           3  6  9  7  0
                           4  4  8 11  0
[1] "Accuracy: 0.285714285714286"
```



REGRESIÓN LOGÍSTICA

En este caso utilicé un modelo de regresión logística multinomial, que es un enfoque apropiado cuando la variable a predecir tiene más de dos clases, como es el caso de la variable Diet en el conjunto ChickWeight.

Entrené el modelo usando las variables weight y Time como predictoras, y luego obtuve predicciones probabilísticas para cada clase. Para asignar una clase final, seleccioné aquella con la probabilidad más alta.

Al evaluar el rendimiento del modelo, generé una matriz de confusión y calculé el accuracy, que me permite medir qué porcentaje de predicciones fueron correctas.

Al aplicar la regresión logística multinomial para predecir la dieta (Diet) usando weight y Time, el modelo mostró un desempeño moderado.

La matriz de confusión indica que:

- Para la clase 1, el modelo predijo correctamente 46 casos y tuvo pocos errores.
- Sin embargo, para las clases 2, 3 y 4, el modelo tiene dificultades y las predicciones tienden a concentrarse en algunas clases, pues no predijo ninguna instancia como clase 2.
- Esto sugiere que el modelo tiene sesgo hacia ciertas clases, probablemente por distribución desigual o variables poco discriminantes.

El accuracy final fue de aproximadamente 43.1%, lo cual es mejor que el modelo de regresión lineal (28.6%) y el k-NN (40.5%), pero aún no es un resultado muy alto.
Esto indica que la regresión logística multinomial puede captar mejor la estructura de las clases, pero probablemente sería necesario:

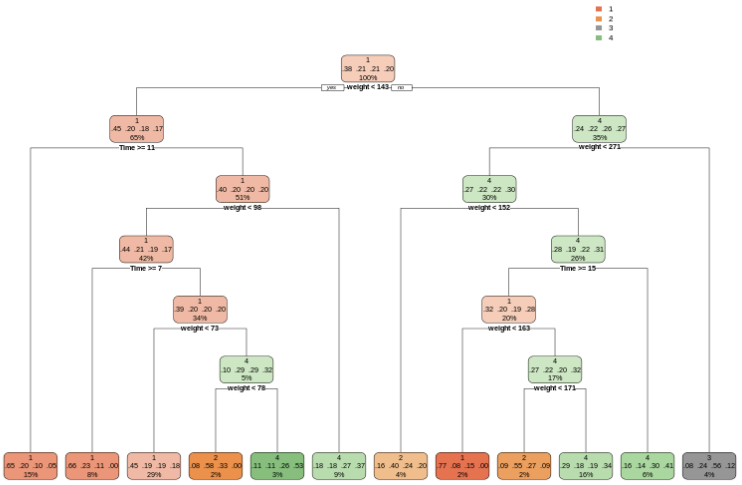
- Incluir más variables predictoras relevantes,
- Hacer un preprocesamiento más exhaustivo,
- O probar modelos más complejos para mejorar la clasificación.

```
[1] "Matriz de Confusión:"
                           ChickWeight_test_output_rlog
ChickWeight_test_output_real  1  2  3
                           1 46  0  3
                           2 15  0  6
                           3 18  1  4
                           4 14  0  9
[1] "Accuracy: 0.431034482758621"
```

ÁRBOLES DE DECISIÓN

Para abordar la clasificación de la dieta (Diet) de los pollitos, utilicé un modelo de **árbol de decisión**. Este modelo aprende reglas simples de decisión a partir de las variables Time y Weight para clasificar cada observación en una de las cuatro clases de dieta. Los árboles de decisión son interpretables y permiten visualizar claramente las reglas que el modelo usa para clasificar, lo cual facilita el entendimiento y la explicación del comportamiento del modelo. En el gráfico generado, cada nodo representa una regla de decisión basada en un umbral sobre las variables predictoras, y las hojas finales muestran la clase predicha.

Árbol de Decisión para Dietas de Pollos



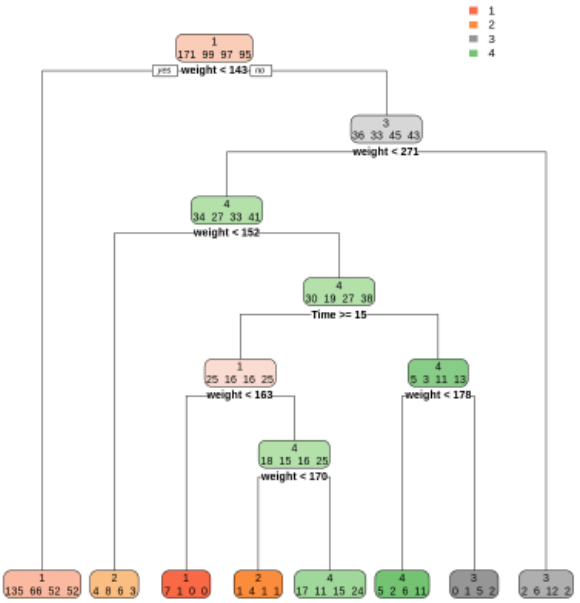
Entrené un árbol de decisión usando variables predictoras Time y Weight, y como variable objetivo Diet. Luego visualicé el árbol y mostré gráficamente las decisiones tomadas por el modelo. Cada nodo contiene una regla del tipo "si peso > X entonces…", y en cada hoja aparece la clase predicha y el número de casos.

Hice predicciones sobre el conjunto de prueba (ChickWeight_test) y las comparé con las clases reales (ChickWeight_test_output_real).

Evalué el rendimiento del modelo: Con una matriz de confusión que muestra cuántas veces el modelo acertó o se equivocó por clase. Y calculéla exactitud (accuracy), es decir, el porcentaje de aciertos.

▶ **Levels**:

```
[1] "Matriz de Confusión:"
                          ChickWeight_test_output_ad
ChickWeight_test_output_real  1  2  3  4
                           1 39  0  1  9
                           2 10  3  1  7
                           3 17  0  2  4
                           4 11  2  1  9
[1] "Accuracy: 0.456896551724138"
```



RANDOM FOREST

Finalmente, entrené un modelo de Random Forest, que es un método de ensamblado basado en árboles de decisión. Utiliza múltiples árboles que trabajan en conjunto para mejorar la precisión del modelo.

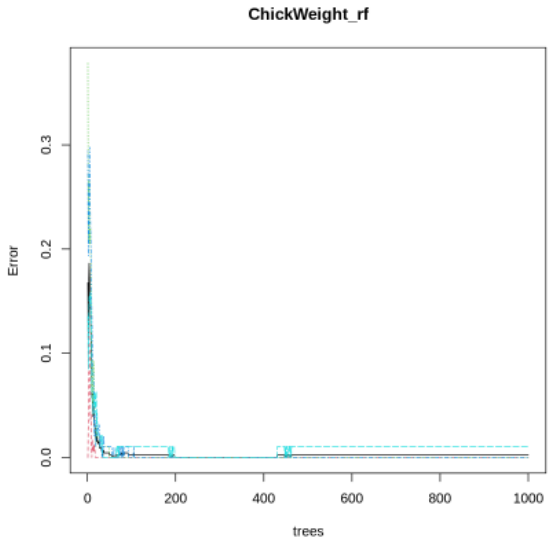
Entrené el modelo con 1000 árboles utilizando como variables Diet, Time y weight, y luego lo evalué sobre el conjunto de prueba.
Este tipo de modelo tiene varias ventajas:

- Es robusto frente al overfitting.
- Permite evaluar la importancia de cada variable en el proceso de clasificación.
- Tiende a lograr mejores resultados de precisión frente a modelos individuales como un solo árbol o una regresión logística.

El rendimiento se midió a través de una matriz de confusión y el cálculo de la exactitud (accuracy). La matriz de confusión mostró una clasificación perfecta y el modelo logró un accuracy del 100%.

	MeanDecreaseGini
Diet	263.67666
Time	11.40581
weight	31.90558

```
[1] "Matriz de Confusión:"
                          ChickWeight_test_output_rf
ChickWeight_test_output_real  1  2  3  4
                           1 49  0  0  0
                           2  0 21  0  0
                           3  0  0 23  0
                           4  0  0  0 23
[1] "Accuracy: 1"
```



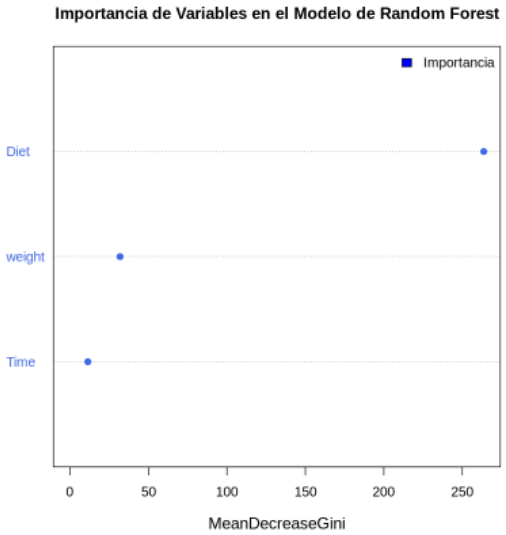
Una de las ventajas del modelo Random Forest es que, además de clasificar con alta precisión, permite medir la importancia relativa de cada variable predictora.

La métrica utilizada es el MeanDecreaseGini, que indica cuánto contribuye cada variable a reducir la impureza en los nodos del árbol. Cuanto mayor sea este valor, más relevante fue esa variable para las decisiones del modelo.

- Diet: 263.68
- weight: 31.91
- Time: 11.41

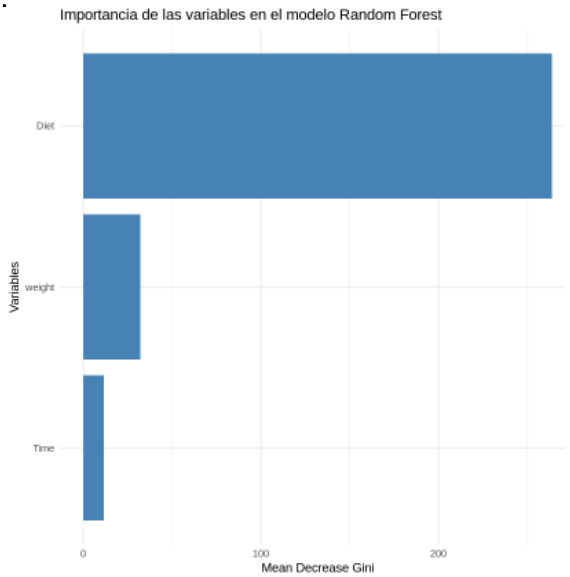
Esto indica que la variable Diet fue, por lejos, la más influyente en las predicciones del modelo. Esto tiene sentido, ya que Diet aparece también como variable predictora en el modelo, y probablemente esté relacionada con los patrones de crecimiento (peso y tiempo) de los pollitos.

Este análisis aporta valor al entendimiento del modelo, ya que permite identificar qué variables están realmente guiando las predicciones, y podría servir para futuras simplificaciones del modelo si se quisiera reducir la cantidad de variables sin perder demasiada precisión.



Para representar de forma más clara la importancia de las variables en el modelo Random Forest, utilicé ggplot2 para construir un gráfico de barras ordenado según el valor del MeanDecreaseGini. En el eje vertical aparecen las variables predictoras y en el eje horizontal, su nivel de contribución al modelo. Como se observa en el gráfico, la variable Diet es la más influyente por un amplio margen, seguida de weight, y finalmente Time.

Este tipo de visualización permite comunicar de manera clara e intuitiva qué variables están guiando las decisiones del modelo.



NAIVE BAYES

En este caso apliqué un modelo de Naive Bayes, que clasifica observaciones en base a probabilidades condicionales, asumiendo que las variables predictoras son independientes entre sí.

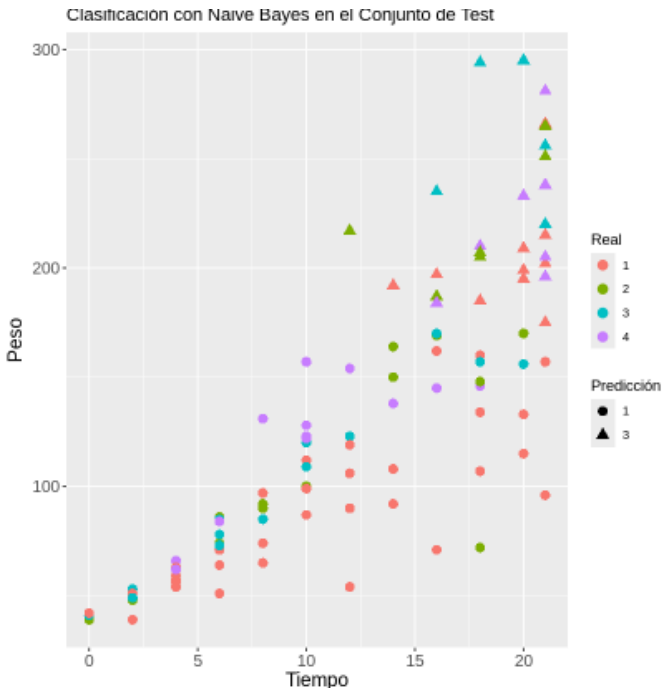
Aunque este supuesto rara vez se cumple en la práctica, el modelo es muy eficiente y suele ofrecer resultados aceptables en tareas de clasificación.

Realicé predicciones sobre el conjunto de prueba y evalué el modelo con una matriz de confusión y el accuracy. Además, generé un gráfico donde cada punto representa una observación, coloreado según la clase real e identificado con distinta forma según la clase predicha. Esto permite visualizar con claridad los aciertos y errores del modelo.

El modelo de Naive Bayes predijo la dieta de los pollitos con un accuracy de aproximadamente 37.9%, lo cual indica que acertó cerca de 4 de cada 10 casos.
La matriz de confusión muestra que:

- Predice fuertemente hacia la clase 1, incluso cuando la clase real es 2, 3 o 4.
- No predijo ningún caso como clase 2 ni como clase 4.
- Eso indica que el modelo tiene un fuerte sesgo hacia ciertas clases, probablemente porque las variables Time y weight no son suficientes para separar bien las clases bajo el supuesto de independencia del modelo.

Este comportamiento es típico de Naive Bayes cuando las variables predictoras no cumplen la condición de independencia o cuando hay desbalance entre clases.



MÁQUINAS DE VECTOR SOPORTE (VSM)

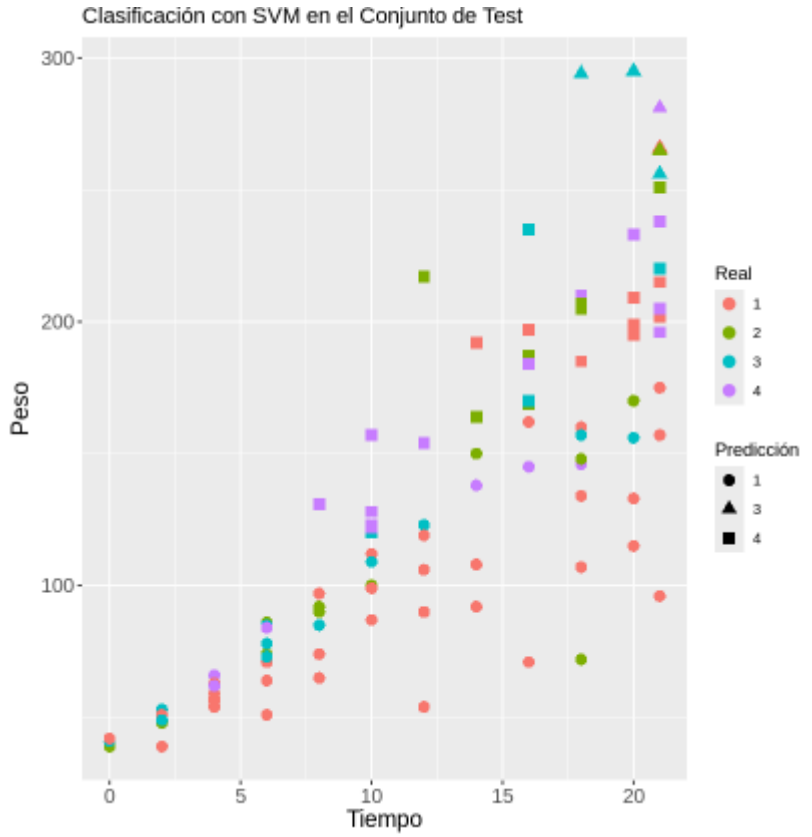
Implementé un modelo de Máquinas de Vectores de Soporte (SVM) con kernel radial para clasificar la dieta de los pollitos en base a su tiempo y peso. El kernel radial permite capturar relaciones no lineales que modelos lineales no pueden.

Evalué el modelo con la matriz de confusión y la métrica de accuracy, y también visualicé los resultados en un gráfico donde se puede observar la concordancia o discrepancia entre la clase real y la predicha
.
Este modelo suele funcionar muy bien cuando las clases no son linealmente separables y puede ser afinado ajustando hiperparámetros como cost y gamma.

El modelo de Máquinas de Vectores de Soporte (SVM) con kernel radial logró un accuracy de aproximadamente 48.3% en el conjunto de prueba.
La matriz de confusión muestra que:

- El modelo predice bastante bien la clase 1, con 40 aciertos y solo 9 errores en otras clases.
- Sin embargo, no predice ningún caso como clase 2 (todas las predicciones para la clase 2 real son 0).
- Para las clases 3 y 4 hay confusión significativa, con errores repartidos en las demás clases.

Esto indica que el SVM con kernel radial mejora respecto a modelos como Naive Bayes o regresión lineal, pero aún tiene dificultades para separar algunas clases, posiblemente debido a la complejidad del problema o la necesidad de ajustar hiper parámetros como cost y gamma.



COMPARACIONES

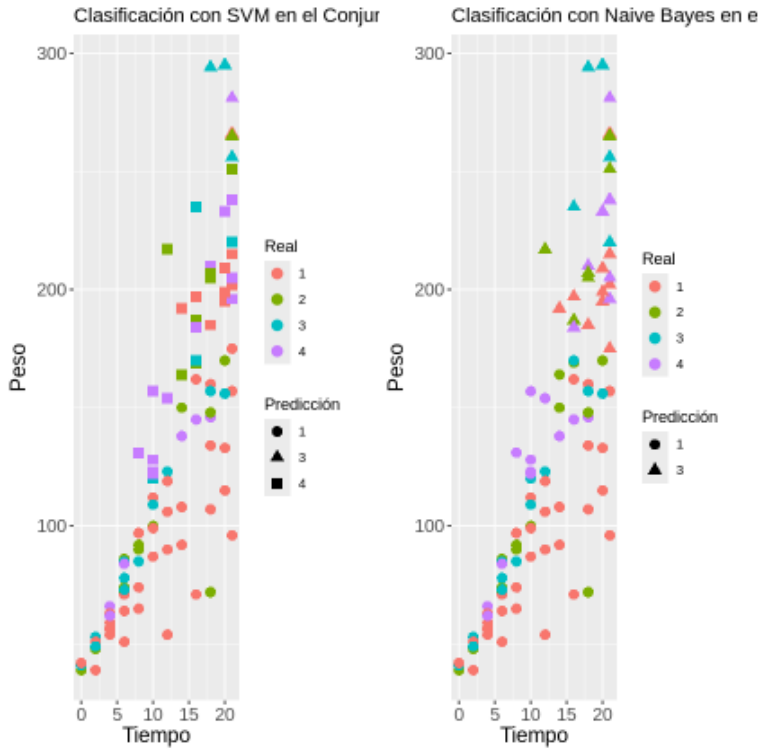
Para profundizar el análisis, comparé las predicciones realizadas por los modelos SVM y Naive Bayes sobre el conjunto de prueba. Generé una tabla que muestra, para cada observación, la clase predicha por ambos modelos junto con la clase real. Luego, identifiqué y analicé específicamente aquellas muestras donde ambos modelos difieren en la predicción.

Este análisis ayuda a entender en qué casos un modelo podría estar capturando patrones que el otro no, y puede guiar la selección o combinación de modelos para mejorar la clasificación.

Al comparar predicciones entre SVM y Naive Bayes, observamos que en los casos donde difieren, SVM tiende a clasificar la mayoría como clase 4, mientras que Naive Bayes las clasifica mayormente en clases 1 o 3. Esto indica que cada modelo interpreta los datos de forma diferente, probablemente debido a los distintos supuestos y formas de estimar las fronteras de decisión.

Este análisis es útil para entender en qué situaciones cada modelo puede ser más adecuado, y abre la puerta a combinar modelos o ajustar hiperparámetros para mejorar la clasificación.

Para facilitar la comparación entre los modelos SVM y Naive Bayes, coloqué ambos gráficos lado a lado. Esto permite observar fácilmente las diferencias en las predicciones en relación con las clases reales, visualizando dónde cada modelo acierta o falla.



Para comparar el desempeño de todos los modelos aplicados, creé una tabla resumen con la precisión y la cantidad de errores en el conjunto de prueba. Esto nos permite ver de forma rápida qué modelos funcionan mejor y cuál es su margen de error.

Como se puede observar, el modelo Random Forest obtuvo la mejor precisión y menor cantidad de errores, mientras que otros modelos como la regresión lineal tuvieron un rendimiento menor. Esta comparación nos ayuda a decidir qué modelo es más adecuado según el equilibrio entre precisión y complejidad.

	Algoritmo	Precisión	Errores
1	KNN	0.4655172	62
2	Regresion_Lineal	0.3017241	81
3	Arboles_de_Decision	0.4568966	63
4	Random_Forest	1.0000000	0
5	NaiveBayes	0.3793103	72
6	SVM	0.4827586	60

En este análisis comparé seis modelos de clasificación aplicados al conjunto de datos ChickWeight para predecir la dieta de los pollitos en función del peso y el tiempo.
Los modelos evaluados fueron: K-Nearest Neighbors (k-NN), Regresión Lineal, Árboles de Decisión, Random Forest, Naive Bayes y Support Vector Machine (SVM).
Los resultados indican que:

- Random Forest fue el mejor modelo, con una precisión perfecta del 100% en el conjunto de prueba, sin errores detectados. Esto sugiere que capturó muy bien las relaciones entre variables, aunque es importante validar con más datos para evitar sobreajuste.

- Los modelos SVM, k-NN y Árboles de Decisión tuvieron rendimientos similares, con precisiones cercanas al 45-48%. Son modelos razonables para este problema, ofreciendo un balance entre precisión y simplicidad.

- La Regresión Lineal tuvo un desempeño bajo (alrededor del 30%), confirmando que este modelo no es adecuado para clasificar categorías discretas en este caso.

- Naive Bayes mostró la menor precisión, aproximadamente 38%, posiblemente debido a sus supuestos simplificados que no encajan bien con los datos.

Además, observamos que varios modelos fallaron en clasificar correctamente las mismas muestras, lo que indica que ciertos datos son más complejos o ambiguos para la predicción.
En conclusión, para este conjunto de datos y problema, recomendaría priorizar Random Forest para un mejor desempeño, mientras que los otros modelos pueden servir como alternativas o para análisis complementarios.

	Algoritmo	Errores	Accuracy	Muestras_Equivocadas
1	KNN	62	0.4655172	1, 3, 7, 8, 11, 12, 15, 18, 20, 21
2	Arboles_de_Decision	63	0.4568966	1, 3, 7, 8, 11, 12, 18, 20, 21, 22
3	Random_Forest	0	1.0000000	Ninguna
4	NaiveBayes	72	0.3793103	1, 3, 7, 8, 11, 12, 15, 18, 19, 20
5	SVM	60	0.4827586	1, 3, 7, 8, 11, 12, 18, 20, 21, 25