|  |
| --- |
| Resultado de imagen para machine learning |
| Aprendizaje Automático  Arboles de decisión |
| |  |  |  | | --- | --- | --- | | Raúl Carlomagno |  |  | |

Contenido

[Introducción 2](#_Toc514444747)

[Objetivos 3](#_Toc514444748)

[Dataset 4](#_Toc514444749)

[Procedimiento 5](#_Toc514444750)

[Bateria de pruebas 5](#_Toc514444751)

[Sobreajuste y poda (CF) 5](#_Toc514444752)

[Precisión: 6](#_Toc514444753)

[Tamaño del árbol 6](#_Toc514444754)

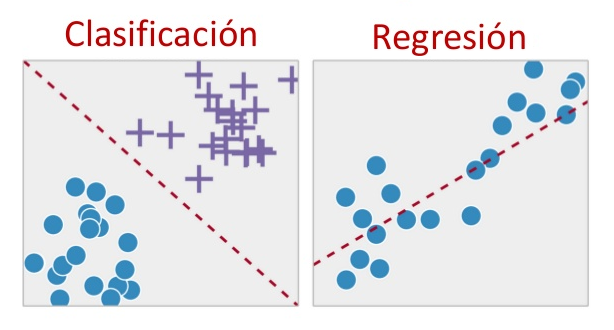
[Sobreajuste y poda (minNumObj) 6](#_Toc514444755)

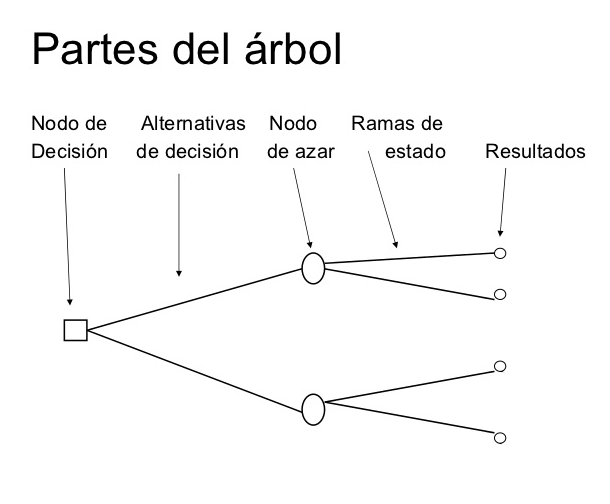
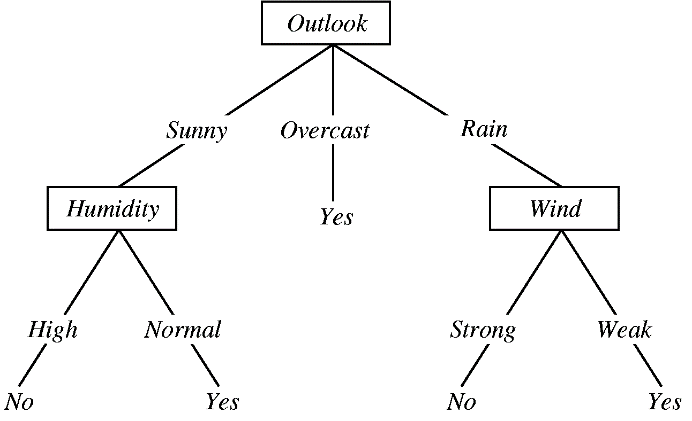
[Precisión 7](#_Toc514444756)

[Tamaño de árbol 7](#_Toc514444757)

# Introducción

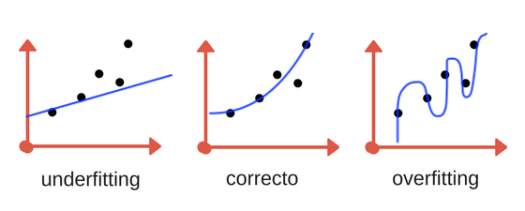
El Machine Learning o Aprendizaje Automático es una rama de la inteligencia artificial que nace durante la segunda mitad del siglo XX. Su principal objetivo es desarrollar métodos capaces de generalizar comportamientos y reconocer patrones a partir de la información de entrada, permitiendo que computadoras encuentren información y se comporten de cierta forma sin que hayan sido explícitamente programadas para ello.  
  
Existen principalmente 3 categorias de algoritmos de aprendizaje automático o machine learning: aprendizaje supervisado (supervised learning), aprendizaje no supervisado (unsupervised learning) y aprendizaje por refuerzo (reinforcement learning).  
Los algoritmos de aprendizaje supervisado tienen como objetivo identificar un modelo a partir de datos de entrenamiento en los cuales se conoce el valor verdadero de salida, y usar este valor para predecir datos futuros o desconocidos.  
El aprendizaje supervisado tiene 2 tipos de actividad como resultado de la aplicación de un algoritmo de machine learning. Las 2 actividades pueden ser de clasificación o de regresión.  
La clasificación tiene como objetivo predecir etiquetas o clases. Estas etiquetas son valores discretos no ordenados que pueden ser entendidos como categorías. Un ejemplo son los filtros de spam, donde a partir de un set de correos electrónicos marcados correctamente como SPAM / NO-SPAM, se entrena un modelo para determinar qué etiqueta se debe asignar a un próximo correo revisando el valor/contenido de sus atributos (contenido, asunto, remitente, etc.).  
Por otro lado, la regresión, se utilizan para la predicción de valores continuos. En el análisis de regresiones, se entrega un número o números (predictor) que explican el valor de salida, es decir, los valores de entrada guardan relación con la variable a explicar. Dentro de las regresiones, la más común es la regresión lineal, que mediante una ecuación de primer orden busca explicar un valor de salida. Un ejemplo de regresión podría ser predecir el volumen de ventas de un próximo periodo para un sitio de e-commerce.



En el presente informe se utilizará un algoritmo perteneciente a la categoría de aprendizaje supervisado y se realizará una clasificación a través del mismo, hay versiones del algoritmo que también soporta tareas de regresión.  
El algoritmo en cuestión, se basa en la **inducción** a partir de la creación de árboles de decisión y es conocido simplemente como arboles de decisión, decision trees o IDT (induction decision trees).  
Muy resumidamente, un árbol puede ser "aprendido" mediante el fraccionamiento del conjunto inicial en subconjuntos basados en una prueba de valor de atributo. Este proceso se repite en cada subconjunto derivado de una manera recursiva llamada particionamiento recursivo. La recursividad termina cuando el subconjunto en un nodo tiene todo el mismo valor de la variable objetivo, o cuando la partición ya no agrega valor a las predicciones.  
En el presente trabajo se utilizará el algoritmo de árbol de decisión J48, es una implementación que forma parte de una herramienta de datamining open source llamada Weka.  
MAS DETALLES Y LIMITACIONES DEL ALGORTIMO   
CAMBIAR GRAFICOS  
 

# Objetivos

Todo algoritmo tiene sus fortalezas y debilidades, no hay un algoritmo perfecto que entregue excelentes modelos sin importar el problema a resolver. El presente trabajo tiene como objetivo someter al algoritmo J48 a una serie de entrenamientos cuyos resultados puedan ser evaluados y asi poder realizar un análisis sobre el comportamiento del mismo.  
Analizando los resultados de las pruebas de validación y testing, podremos sacar conclusiones sobre como actua en determinadas situaciones según el tipo de dataset que le estemos ingestando en el entrenamiento y como actua contra las diversidades.  
Existen 2 cuestiones con los algoritmos de aprendizaje supervisado: overfitting (sobreajuste) y underfitting (subajuste) de los datos. El overfitting y el underfitting son los responsables de obtener malos resultados. Cuando entrenamos un modelo intentamos “hacer encajar” (fit en inglés) los datos de entrada entre ellos y con la salida. El fallo común de nuestro modelo generado es de generalizar (“encajar”) el conocimiento que pretendemos que adquieran.  
El underfitting se podría explicar al entrenar un modelo para el reconocimiento de razas de perros, pero entrenarlo con una sola raza de perro. Cuando se intente reconocer una raza nueva, el modelo fallará en el reconocimiento por falta de suficientes muestras; no pudo generalizar el conocimiento.  
Por otra parte, el overfitting se puede ejemplificar siguiendo el ejemplo anterior, al entrenar un modelo para el reconocimiento de razas de perros, pero entrenarlo con 10 razas de perros de color marrón. Cuando se quiera reconocer una raza de perro color blanco, el modelo fallará porque no tiene estrictamente los mismos valores de las muestras de entrenamiento (entrenó a la perfección solamente razas de perros marrones).  
El algoritmo debería ser capaz de generalizar conceptos y que al intentar predecir una nueva muestra con algunos datos desconocidos, sea capaz de dar una clasificación fiable dependiendo del grado de generalización que haya tenido en su entrenamiento.



Deberemos encontrar un punto medio en el aprendizaje de nuestro modelo en el que no estemos incurriendo en underfitting y tampoco en overfitting.  
Veremos como se comporta el algoritmo al inducir ruido en alguna variable y si es capaz de manejar cuestiones como el underfitting y overfitting.

# Dataset

Para las pruebas se utilizará un dataset de la plataforma de crowdfunding (financiamiento colectivo) Kickstarter.  
Kickstarter funciona facilitando la captación de recursos monetarios del público en general, un modelo que evita muchas vías tradicionales de inversión. Los proyectos pertenecen a una categoría determinada y deben cumplir con las directrices de Kickstarter para ponerse en marcha (proyectos de caridad, de causas, de “financiación de vida” y recaudación de fondos sin límites fijos no están permitidos). Los dueños del proyecto eligen una fecha límite y un mínimo objetivo de fondos a recaudar. Si el objetivo monetario elegido no es recolectado en el plazo, el dueño del proyecto no percibe los fondos y el proyecto no se financia.

El dataset fue descargado desde <https://www.kaggle.com/kemical/kickstarter-projects/data>  
El nombre del archivo utilizado es *“ks-projects-201801.csv”*, pero este no fue el dataset final que se utilizó para el informe, sino, que se hizo un trabajo de pre procesamiento sobre el mismo.  
Del dataset original se descartaron ciertos atributos o variables, por ejemplo, el ID del proyecto, el titulo, la cantidad de depositantes, el dinero finalmente recaudado (para otro trabajo, se podría llegar a realizar un aprendizaje supervisado de regresión sobre este atributo), etc. Otros atributos sufrieron conversiones, por ejemplo, el estado del proyecto era un atributo categorico con 4 valores, que finalmente, luego de analizar los estados disponibles, fue nuestra variable target binaria. Los proyectos al ser publicados en distintos países, poseían su objetivo a recaudar en distintas monedas, teniendo esto en cuenta, se establecio el objetivo a recaudar en dólares convirtiendo cada valor a dólares según el tipo de cambio actual.  
El dataset disponía aproximadamente de 370.000 muestras, las cuales pueden llegar a ser útiles para entrenar un modelo de la vida real, pero en cambio, para realizar el presente trabajo, semejante cantidad de muestras iba a retrasar la experiencia, ya que vamos a realizar una cantidad importante de entrenamientos y si se utilizaría la cantidad de muestras originales, llevaría una cantidad de tiempo innecesaria. Es por eso que a partir del pre procesamiento realizado nuestro dataset final para experimentar fue de 6.000 muestras representativas. Haciendo énfasis en representativas, ya que el dataset original estaba desbalanceado para Estados Unidos, el 80% de las muestras correspondían a ese país; entonces la muestra tomada fue más homogénea teniendo en cuenta la variable país.  
  
Estructura final del dataset:

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Valores posibles | Descripción |
| main\_category | categórico | 15 | Categoría del proyecto |
| country | categórico | 20 | País del proyecto |
| days\_funding | numérico entero | 7 a 62 | Plazo de días para llegar al objetivo de dinero a recaudar. (desde la creación del proyecto) |
| year\_launched | categórico | 9 (2009 a 2017) | Año de lanzamiento del proyecto |
| month\_launched | categórico | 12 (January a December) | Mes de lanzamiento del proyecto |
| usd\_goal | numérico entero | 1000 a 50000 | Objetivo monetario a recaudar en dolares |
| funded | binario | yes / no | Valor binario que determina si el proyecto llegó al objetivo monetario de financiamiento o no |

El trabajo de procesamiento se encuentra disponible en el script *“preprocess.R”*

# Procedimiento

Se realizarán 5 tipos de pruebas contra el algoritmo, cada una con pruebas iterativas de entrenamiento variando diferentes parámetros del algoritmo o con modificaciones en el dataset ingestado para el entrenamiento. Las 2 primeras consisten en analizar el comportamiento de J48 con respecto a la poda del árbol y el overfitting, variando ciertos parametros. Otra de las pruebas es analizar al algoritmo respecto a los datos faltantes, también se analizará su comportamiento respecto a la tolerancia al ruido y la discretizacion de datos numéricos.

El dataset de 6.000 muestras fue fraccionado en 2 partes, una para training (80%, *“ks-projects-processed-train.csv”*) y otra para testing (20%, *“ks-projects-processed-test.csv”*). Quedando finalmente 4.800 y 1.200 muestras respectivamente.

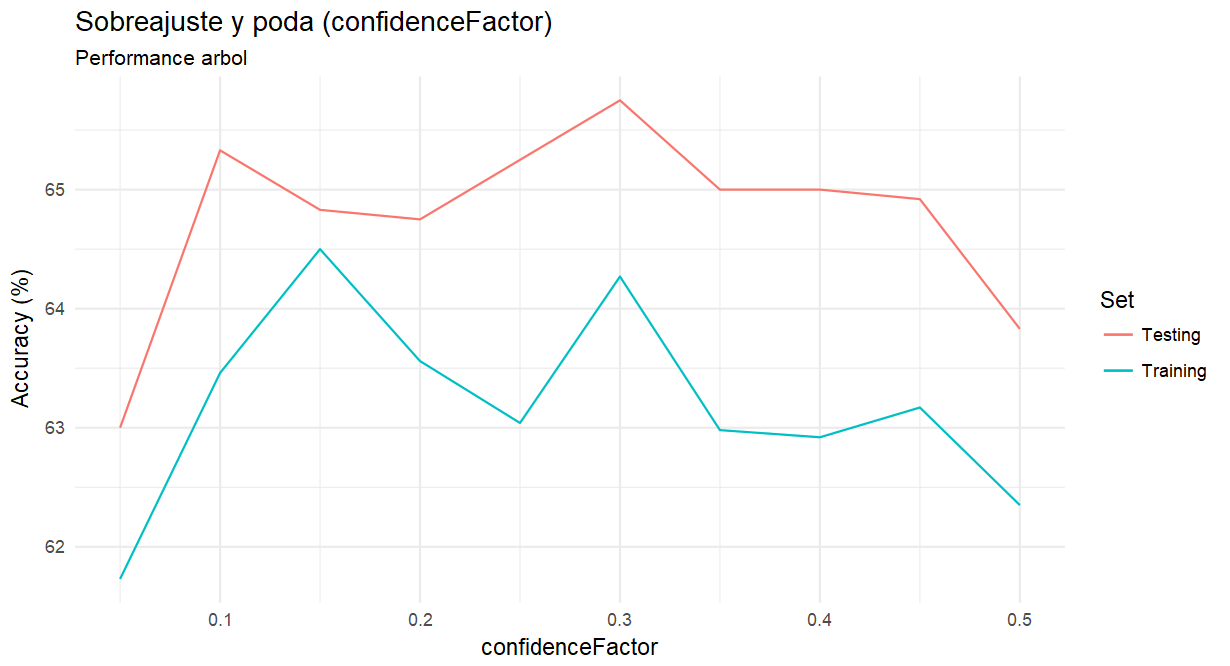
Como se especificó en un punto anterior, se utilizó el algoritmo J48 de Weka, pero no se utilizó directamente desde Weka, sino que se realizó un script (“trainValidateWeka.R”) que realiza los entrenamientos dinámicamente (modificando parámetros del algoritmo o alterando el dataset) y genera los resultados para que luego otro script (*“plots.R”*) genere los plots para realizar los diversos análisis.

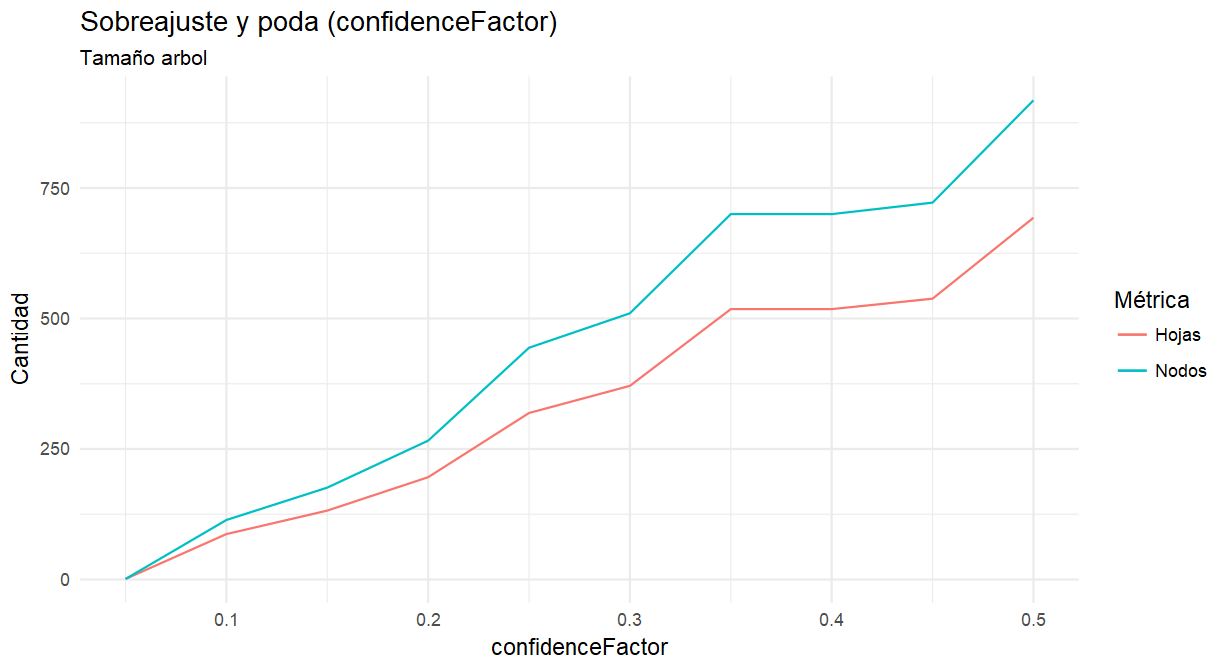
# Bateria de pruebas

## Sobreajuste y poda (CF)

En esta prueba se somete al algoritmo a diversos entrenamientos variando un parámetro llamado Confidence Factor (CF, factor de confianza) desde el valor 0,05 hasta 0,5 en pasos de a 0,05. Por lo tanto se ejecutaron 10 entrenamientos donde se observa la performance en training y testing, asi como también el tamaño final del árbol.  
El parámetro influye en la poda del árbol, cuanto más bajo es el valor, mas poda el algoritmo al arbol. Depende de este valor que el modelo quede sobreajustado o subajustado, hay que encontrar un balance.

Precisión:   
Se puede observar un fenomeno poco común que puede llegar a estar relacionado con la naturaleza de los datos, la precisión es mayor en el set de testing. Sin embargo ocurre algo esperado, cuanto más poda el algoritmo (más bajo el CF), mas se reduce la precisión, hasta un punto en donde la precisión vuelve a decrecer. Vemos como el comportamiento en los ejes de ambos set es similiar, parece q uno copia al otro a nivel visual. Weka por defecto entrena con un valor de 0,2 para el CF, que en nuestro caso en testing estaría acertando con una precisión de 65% aproximadamente. Sin embargo se observa que con un CF de 0,3 obtenemos la mejor precisión para nuestro dataset en testing, un 65,6% aproximadamente. Los desarrolladores del algoritmo J48 deben haber establecido por defecto al CF en 0,2 debido a que en distintas pruebas, contextos y entornos; habrán obtenido muy buenos resultados para la poda del árbol. Un modelo resultante que no se encuentra afectado por el overfitting ni por el underfitting.

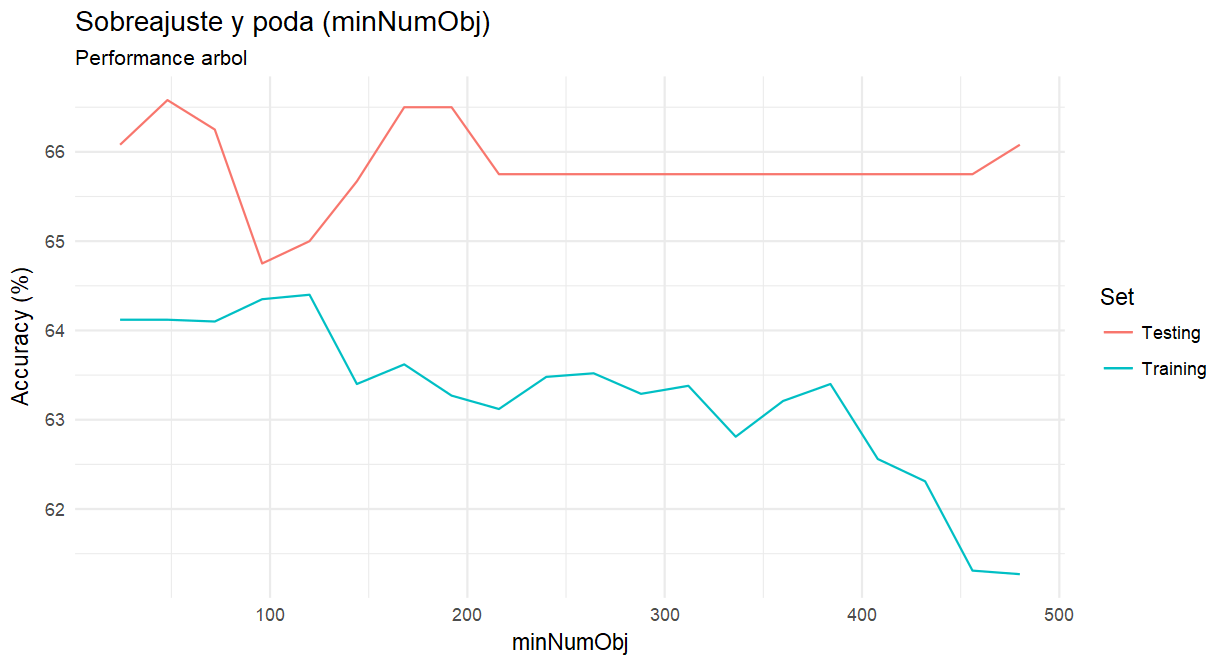
Tamaño del árbol:  
Como se detalló en el párrafo anterior, el valor del CF es inversamente proporcional al tamaño del árbol, cuanto más chico es CF, más poda el algoritmo y por ende el tamaño del árbol es inferior. Cuando se habla de tamaño se hace referencia a la cantidad de hojas y de nodos que forman el árbol. Se observa un hecho peculiar, el CF entre 0,35 y 0,45 no genera variaciones en el tamaño del árbol, puede llegar a ser porque al intentar podar el algoritmo no justifica esa poda para intentar generalizar.  
Dejando de lado esa observación se observa un comportamiento lineal entre el tamaño del árbol y el valor del CF.

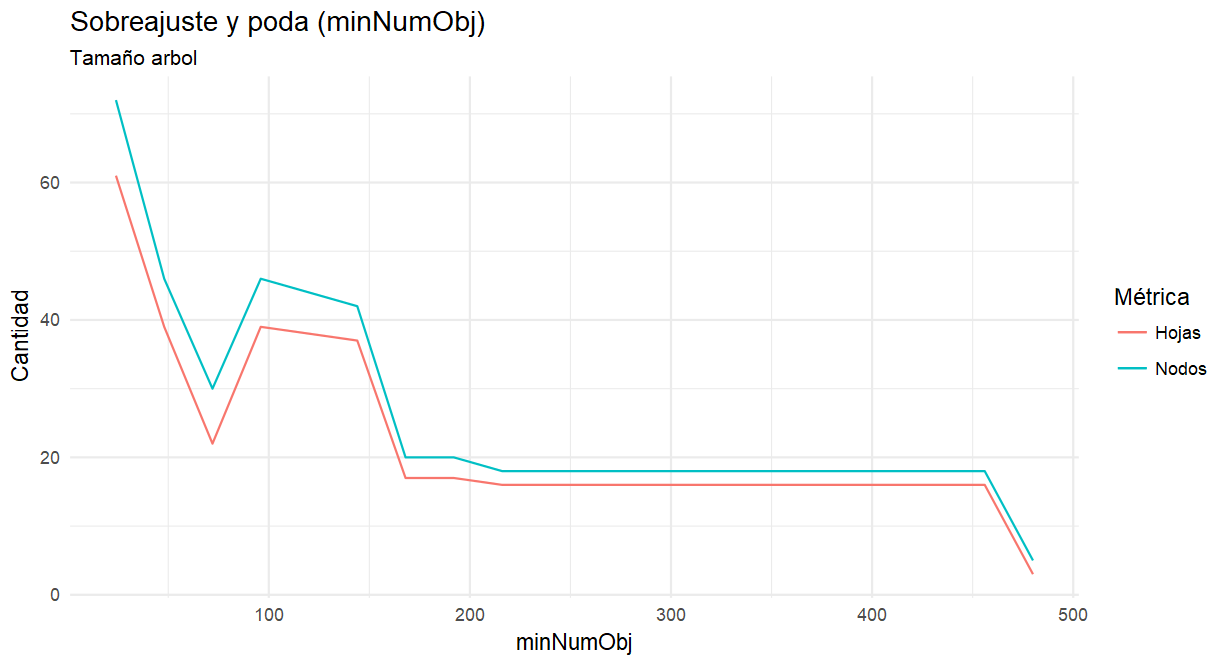


## Sobreajuste y poda (minNumObj)

Al momento de la poda, es posible especificarle al algoritmo otro parámetro que impacta directo en el proceso de poda. Este parámetro es la mínima cantidad de objetos (minNumObj) que debe poseer un nodo, si el nodo no llega a esa cantidad de objetos minimos, ese nodo se poda y los objetos que estaban en ese nodo son trasladados al nodo superior, alli se evalua nuevamente la cantidad minima de objetos de ese nodo y asi sucesivamente hasta que esa rama llega a obtener un nodo que supere la cantidad minima de objetos. Es por eso que en esta prueba se ejecutaron una cantidad de entrenamientos variando este parámetro, el parámetro va desde 0,5% hasta el 10% de la cantidad de muestras del dataset de training, en pasos de 0,5%. Por lo tanto, para nuestro caso se empezaría con 24 objetos minimos para cada nodo, hasta 480 objetos minimos, aumentando de a 24 objetos mínimos.

Precisión:  
En el set de testing, se observa un comportamiento pseudo estable respecto a la precisión, se mantiene entre 65% y 66,5% aproximadamente y casi constante, más que nada entre los 200 y 450 objetos minimos. Por ende se puede concluir que con datos de “la vida real” (testing) la poda no afectó la precisión en medidas significativas, es mas, con 480 objetos minimos (árbol muy podado) la precisión fue buena y la precisión mas baja se ve con 100 objetos minimos, el árbol fue poco podado y aun asi obtuvo un poco menos de 65% de precisión.  
Por otra parte, en el set de training a medida que el árbol era más podado (aumentan la cantidad de objetos minimos por nodos), la precisión se fue reduciendo, esto se debe a que el árbol con poca poda aprende casi de memoria y obtiene buena precisión, pero a medida que va generalizando su precisión decrece en favor de aumentar en testing.



Tamaño de árbol:  
Nuevamente se observa un comportamiento que no responde a lo esperado, que sería que a medida que la cantidad minima de objetos crece (aumenta la poda) el tamaño del árbol decrece, un comportamiento lineal inverso o inversamente proporcional. Pero esto no ocurre suavemente en el gráfico, sino que se observan 2 situaciones interesantes. Si se compara el inicio del eje x (minNumbObj) con el final se ve claramente que el tamaño del árbol se redujo, pero no de forma lineal. Cuando la cantidad de objetos minimos es de 75 el tamaño del árbol comienza a aumentar en vez de reducirse, hasta que la cantidad es 100 objetos y vuelve suavemente a reducir su cantidad. A los 150 objetos el tamaño del árbol cae considerablemente y entra en una meseta hasta los 460 objetos, se podría deducir que el árbol ya había llegado a una poda bastante y no era posible seguir siendo podado. Pero luego cuando se proponen 470 objetos minimos, el árbol es podado considerablemente, quedando con 6 nodos y 4 hojas aproximadamente.