Bootstrapping y aplicaciones al Machine Learning

Carrascal Ibañez Juan David[a], Forero Laiton Angie Daniela[b], Mora Ramirez David Felipe[c,*]

[a] *jdcarrascali@unal.edu.co*

[b] *anforerol@unal.edu.co*

[c] *dmorara@unal.edu.co*

Abstract

En este articulo se trabajará la teoría y algunas aplicaciones en el Machine Learning de un método de remuestreo muy conocido, el Bootstrapping, para ello se fundamentarán ciertos conceptos relacionados con algunas ramas de las matemáticas y posteriormente nos enfocaremos en explicar una variante del Bootstrapping que se conoce como Bagging y que se posiciona como uno de los mejores algoritmos de aprendizaje estadístico, dado que este es muy potente y robusto, para dicho método generaremos múltiples versiones que predicen cierto comportamiento o resultado numérico y que se usaran para obtener una nueva versión que ofrecerá ganancias sustanciales en la precisión.

Keywords: Bootstrapping, Bagging, Arboles de Decisión, Remuestreo, modelamiento.

1. Introducción

El *bootstrapping*, propuesto Brandley Efron a inicios del siglo XX, es una herramienta estadística que comúnmente se utiliza para estimar paramétros desconocidos de una variable aleatoria a partir de generar un nuevo conjunto de datos a partir de un proceso de remuestreo.Esto en la practica es posible gracias a los métodos de simulación de Monte Carlo y a la existencia de computadores lo suficientemente potentes para ejecutar los algoritmos que este proceso conlleva. En Machine learning, el *bootstrapping* también está presente, dentro de metodología aprendizaje conjunto(*ensemble learning*), donde el objetivo es combinar las predicciones de varios estimadores base construidos con cierto algoritmo de aprendizaje con el fin de mejorar la generalidad y robustez del estimador. En nuestro estudio particular del *bagging* construimos varios estimadores de forma independiente, que serán entrenados en muestras usando *bootstrapping* sobre los datos de entrenamiento, y cuyas predicciones serán promediadas con el fin de obtener un estimadores con una varianza reducida. En particular nos centraremos en el *bagging* de arboles de decisión, ya que son uno de los algoritmos de Machine Learning mas usados por su carácter simple y fácil aplicabilidad. Finalmente, implementaremos la teoría expuesta a lo largo del articulo con un problema de regresión en el que nuestro objetivo es predecir el precio de viviendas a a partir de una base de datos del censo de California de 1990.

2. Conceptos Previos

Se comenzará enunciando algunas definiciones y términos previos relacionados estrechamente con probabilidad y estadística, los cuales serán necesarios para entender la teoría al rededor del Bootstrapping y el bagging.

2.1. Método de Montecarlo:

El método de Montecarlo fue inventado por Stanislaw Ulam y John Von Neumann en 1946, es un método de simulación que genera posibles resultados a problemas de carácter matemático por medio de un muestreo

*Departamento de Matemáticas, Universidad Nacional de Colombia, Bogotá, Colombia

estadístico, es decir se intenta imitar un comportamiento de un sistema real y su conjunto de posibilidades, lo cual permite tener una predicción del comportamiento de las variables que se están estudiando.

Este método posibilita el ensayo de experimentos con remuestreo de números pseudoaleatorios en una computadora, es así como solucionar ecuaciones para las cuales los métodos deterministas son bastante difíciles y arduos de efectuar pueden tener solución mediante la visualización de todas sus posibilidades y sus distintas probabilidades.

2.2. Valor esperado:

El valor esperado de una variable aleatoria es uno de los resultados más importantes y más utilizados, lo conocemos en su forma discreta como promedio, en el caso en el que X es discreta con soporte $D_X = \{x_0, x_1, \cdots, x_n\}$ y función de masa de probabilidad $P(X = x_i)$, se define como

$$E[X] = \sum_{x \in D_X} x \cdot P(X = x)$$

En el caso en el que variable aleatoria es continua y tiene función de densidad $f_X(x)$ el valor esperado se define como $E[X] = \int_\Omega x \cdot f_X(x)dx$.

Algunas propiedades importantes del valor esperado son:

$$E[c] = c \quad \text{(Siendo } c \text{ una constante)}$$
$$E[a + bX] = a + bE[X] \quad \text{(El valor esperado se comporta linealmente)}$$

2.3. Varianza:

La varianza de una variable aleatoria es una medida de dispersión definida como se sigue, sea X una variable aleatoria con valor esperado $\mu = E[X]$, definimos la varianza de la variable aleatoria X como $Var(X) = E[(X - \mu)^2]$, lo cual se puede representar de una manera equivalente como

$$\begin{aligned} Var(X) &= E[(X - \mu)^2] \\ &= E[X^2 - 2\mu X + \mu^2] \\ &= E[X^2] - 2\mu E[X] + \mu^2 \quad \text{(Usando propiedades del valor esperado)} \\ &= E[X^2] - 2\mu^2 + \mu^2 (E[X] = \mu) \\ &= E[X^2] - \mu^2 \end{aligned}$$

Así $Var(X) = E[X^2] - \mu^2$, algunas propiedades importantes de la varianza y que se usarán con frecuencia son:

$$Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$$
$$Var(cX) = c^2 Var(X)$$

2.4. Covarianza:

La covarianza de dos variables aleatorias nos permite conocer como se comportan las variables con respecto a sus medias, por ejemplo ¿Como se comporta una variable cuando la otra disminuye o aumenta?, esta se define de la siguiente manera: Sean X, Y dos variables aleatorias, siempre que los valores esperados de dichas variables aleatorias existan la covarianza entre X y Y esta dada por

$$Cov(X, Y) = E[(X - E[Y])]E[(Y - E[Y])]$$

Por propiedades del valor esperado esta definición es equivalente a $Cov(X, Y) = E[XY] - E[X]E[Y]$, adicionalmente una propiedad importante y que se utilizará de la covarianza es $Cov(cX, Y) = Cov(X, cY) = cCov(X, Y)$.

2.5. Muestra aleatoria:

Se dice que $\{X_i\}_{i\in I}$ es una muestra de variables aleatorias independientes e idénticamente distribuidas si:

1. Para cada $n \in \mathbb{Z}^+$ se dice que $X_1, \cdots, X_n$ son variables aleatorias independientes.
2. Para todo $i, j \in \mathbb{Z}^+$; X_i y X_j tienen la misma distribución.

Adicionalmente, un conjunto de variables aleatorias $X_1, X_2, \cdots, X_n$ se llaman independientes si $P(X_1 \in A_1, X_2 \in A_2, \cdots, X_n \in A_n) = P(X_1 \in A_1)P(X_2 \in A_2)\cdots P(X_n \in A_n)$, siendo $A_1, A_2, \cdots, A_n$ conjuntos de Borel de $\mathbb{R}$.

2.6. Estimador puntual:

Es una estadística de dimensión igual al número de componentes desconocidas de un modelo probabilístico que es asignada a este de manera que queda completamente determinado.

2.7. Sesgo de un estimador:

Dado un estimador $\hat{\theta}(X)$ se define su sesgo como:

$$B_\theta(\hat{\theta}(X)) := E[\theta(X)] - \theta$$

2.8. Error cuadrático medio de un estimador:

- La función de pérdida es una función no negativa que busca evaluar la cercanía de una decisión al verdadero parámetro de acuerdo con alguna medida de distancia.
- La función de riesgo mide la pérdida esperada en todos los escenarios posibles.
- Bajo la función de pérdida cuadrática, la función de riesgo se conoce como error cuadrático medio. Es calculada con la expresión:

$$MSE(\hat{\theta}, \theta) = E_\theta\left[(\hat{\theta} - \theta)^2\right] = B_\theta^2(\hat{\theta}) + Var_\theta(\hat{\theta}(X))$$

3. Fundamentos Teóricos

A continuación se mostrará la teoría al rededor de un método desarrollado por Bradley Efron en 1979 y que dada su filosofía, se puede interpretar como una aplicación del método de Montecarlo, esta técnica es utilizada principalmente para aproximar distribuciones en el muestreo estadístico y posteriormente sera mejorada para desarrollar estimaciones con una mejor varianza.

3.1 Bootstrapping:

Motivación:

Supongamos que queremos invertir una cantidad fija de dinero en dos activos que retornan una ganancia de X y Y respectivamente, donde X y Y son cantidades aleatorias.

Sean $Var(X) = \sigma_X^2, Var(Y) = \sigma_Y^2$ y $Cov(X, Y) = \sigma_{XY}$, queremos encontrar el valor de α que minimiza la expresión $Var(\alpha X + (1-\alpha)Y)$:

$$\begin{aligned} Var(\alpha X + (1-\alpha)Y) &= Var(\alpha X) + Var((1-\alpha)Y) + 2Cov(\alpha X, (1-\alpha)Y) \\ &= \alpha^2 Var(X) + (1-\alpha)^2 Var(Y) + 2\alpha(1-\alpha)Cov(X, Y) \\ &= \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\sigma_{XY}(-\alpha^2 + \alpha) \end{aligned}$$

Derivando e igualando a cero,

$$\begin{aligned} 0 &= \frac{d}{d\alpha} f(\alpha) \\ &= 2\sigma_X^2\alpha + 2\sigma_Y^2(1-\alpha)(-1) + 2\sigma_{XY}(-2\alpha+1) \\ &= (\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})\alpha - \sigma_Y^2 + \sigma_{XY} \end{aligned}$$

De donde obtenemos que: $\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$. Sin embargo, dado que los verdaderos valores de los parámetros que involucran a la expresión de α son desconocidos, para encontrarlos debemos hacer un proceso de estimación de las cantidades $\hat{\sigma}_X^2, \hat{\sigma}_Y^2$ y $\hat{\sigma}_{XY}$, para ello podemos hacer uso de la técnica estadística bootstrapping[**Bootstrappin**

Fundamentos:

El siguiente resultado es clave para la simulación y se puede extender a cualquier tipo de variable aleatoria.

Teorema: Sea X una variable aleatoria absolutamente continua con $F_X(x)$ creciente entonces se tiene que:

$$U = F_x(x) \sim U[0,1]$$

Demostración: Recordemos que la función de distribución de una variable aleatoria está dada por $F_X(x) = P[X \leq x]$. Como $F_X(x)$ es creciente y absolutamente continua entonces $F_X^{-1}(x)$ existe. Así:

$$\begin{aligned} F_U(u) &= P[U \leq u] \\ &= P[F_X^{-1}(u)] \\ &= F_X(F^{-1}(u)) \\ &= u \end{aligned}$$

De esta manera observamos que:

$$F(u) = \begin{cases} 0 & si \quad u < 0 \\ \frac{u-0}{1-0} & si \quad 0 \leq u < 1 \\ 1 & si \quad u \geq 1 \end{cases}$$

Por consiguiente, $F_X(x)$ tiene distribución uniforme con $F_X(x) \sim U[0,1]$.

Función de distribución empírica

Sean $x_1, x_2, ..., x_n$, n datos recogidos de una muestra aleatoria. Se define la función de distribución empírica de los datos como:

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[x_i,\infty)}(x)$$

Es de notar que $F_n : \mathbb{R} \longrightarrow [0,1]$, es no decreciente y no depende del modelo probabilístico[**Bootstrapping2**].

Muestra bootstrap:

Dado un conjunto de datos $\{x_1, x_2, .., x_n\}$ haciendo uso de la distribución empírica le asignaremos a cada valor x_i una probabilidad de $\frac{1}{n}$. A partir de esto, obtenemos un nuevo vector conocido como muestra bootstrap $x^* = (x_1^*, x_2^*, .., , x_n^*)$.

Por otra parte, $\hat{\theta} = s(x^*)$ es una estadística sobre la muestra bootstrap x^*. Aquí $s(\cdot)$ es la misma función que determina al parámetro $s(x)$ en la población. Por ejemplo, si $s(x)$ es la media muestral $\bar{x}$ entonces $\bar{x^*} = \sum_{i=1}^{n} \frac{x_i^*}{n}$.

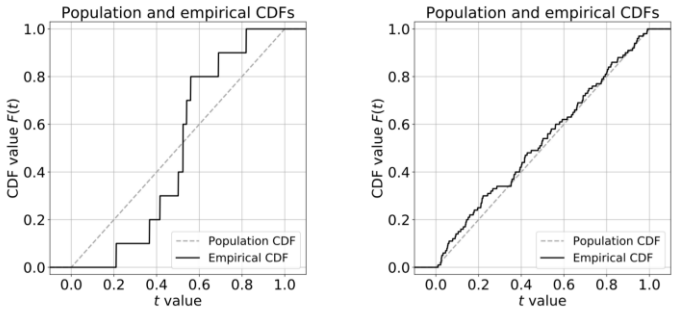
Teorema: Glivenko - Cantelli

Sean $x_1, x_2, ..., x_n$ datos recogidos de una muestra aleatoria $X_1, X_2, .., X_n$ con función de distribución $F_X(x, \theta)$ Entonces:

$$\sup_{x \in \mathbb{R}} |\hat{F}_n(x) - F_X(x, \theta)| \xrightarrow{\text{c.s}} 0 \quad \text{cuando} \quad n \to \infty$$

Es decir, para un tamaño de muestra suficientemente grande, la función de distribución empírica puede ser usada como "aproximación" de la verdadera función de distribución.

Example 2.1. If μ is the uniform distribution on $[0, 1]$, $F_\mu(t) = t$ for $t \in [0, 1]$. Here is a comparison of what the empirical CDF F_{μ_n} may look like compared to F_μ for $n = 10$ and $n = 100$ samples.[1]



Teorema: Simular m valores de $\hat{F}_n(x)$ es equivalente a remuestrear m elementos de $x_1, x_2, ..., x_n$ con reemplazo.

Este teorema inspira el siguiente método

Método de remuestreo:

- Genere B muestras aleatorias de tamaño n,$\{\tilde{x}_{1,i}, \tilde{x}_{2,i}, ..., \tilde{x}_{n,i}\}_{i=1}^{B}$; de la distribución empírica.
- Para cada muestra i,calcule las estadísticas de interés, $T(\tilde{x}_{1,i}, \tilde{x}_{2,i}, ..., \tilde{x}_{n,i})$.
- Estudie las propiedades de las estadísticas de interés, usando las B muestras: $\{T(\tilde{x}_{1,i}, \tilde{x}_{2,i}, ..., \tilde{x}_{n,i})\}$.

Cuando queremos conocer estudiar propiedades de una variable aleatoria resulta muy útil poder generar múltiples muestras a partir de la verdadera distribución de los datos. Sin embargo, en la mayoría de escenarios se desconoce esta distribución y solo se tiene un conjunto de datos $x_1, x_2, .., x_n$ recogidos. En el siguiente ejercicio, ejemplificaremos el funcionamiento del método bootstrap partiendo de un conjunto de datos con distribución conocida.

Generaremos una única muestra aleatoria de tamaño $n = 10$ a partir de números pseudo-aleatorios obtenidos a partir de la distribución $N(\mu = \frac{1}{7}, \sigma^2 = 7)$ con la función *rnorm*

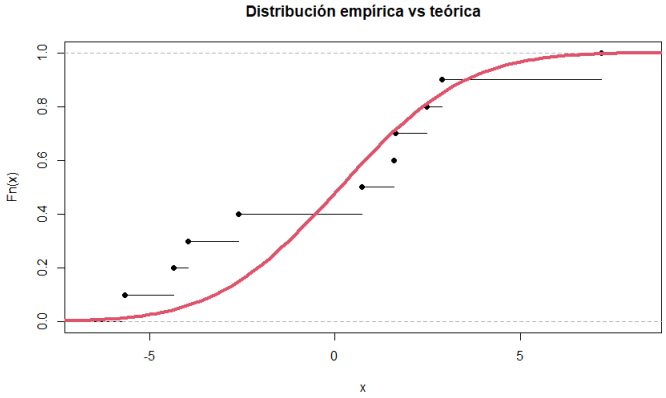
```
data<-rnorm(n=N,mean=(1/7),sd=sqrt(7))
mean(data)
var(data)
var(data)/mean(data)
miu_hat<- 1/7
miu_hat
var_hat<- 7
```

obtenemos la salida

```
> mean(data)
[1] -0.0953547
> var(data)
[1] 4.127261
> miu_hat
[1] 0.1428571
> var_hat<- 7
>
```

A partir de estos datos calculamos la función de distribución empírica con *ecdf* y superponemos la función de distribución teórica.

```
#Distribucion empirica de los datos
y<-ecdf(data)
plot(y,main="Distribucion empirica vs teorica")
curve(pnorm(x, mean = 1/7, sd = sqrt(7)), from = -10,to=10,col=2,lwd=4,add=TRUE)
#Bootstraps
bootstrap<- replicate(n=1000,sample(data,replace=TRUE))
```



De esta manera, observamos que las dos funciones se comportan de una manera muy diferente, a pesar de ello podemos visualizar como es el comportamiento de la sucesión $\sup_{x\in\mathbb{R}} |\hat{F}_n(x) - F_X(x,\theta)|$ en sus valores iniciales.

Haremos un proceso de remuestreo a partir de calcular 1000 muestras bootstrap de la muestra original con la función *sample*:

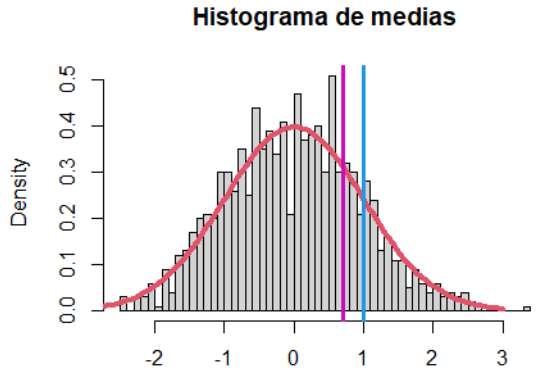
```
#Bootstraps
bootstrap<- replicate(n=1000,sample(data,replace=TRUE))
dim(bootstrap)
[1] 10 1000
```

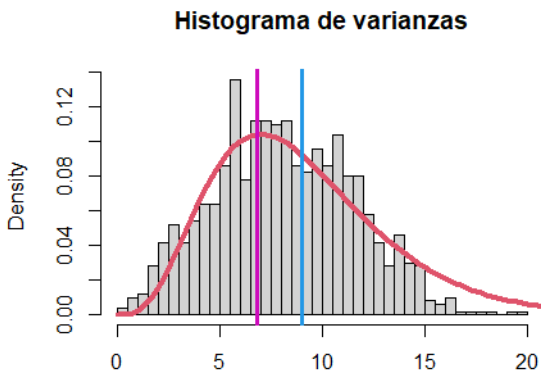
A continuación, procedemos a calcular para cada muestra bootstrap, su media $\left\{\frac{\sqrt{n}(\bar{x_i}-\hat{\mu})}{\hat{\sigma}}\right\}_{i=1}^{B}$ y su varianza $\left\{\frac{(n-1)\hat{s_i}^2}{\hat{\sigma}}\right\}_{i=1}^{B}$ donde $\hat{\mu}$ y $\hat{\sigma}$ son las estimaciones teóricas de los parámetros de la distribución $N(\mu = \frac{1}{7}, \sigma^2 = 7)$. En las siguientes gráficas vamos a representar las estimaciones teóricas con el color magenta y las de la muestra bootstrap con el color azul. De aquí en adelante $B[\hat{\theta}]$ representa el sesgo del estimador θ

```
#Vaianzas
x_i=NULL
c_i<-function(n){
  for (i in 1:N){
    x_i<-append(x_i,bootstrap[i,n])
  }
  return((N-1)*var(x_i)/var(data))
}
#Vector de chis e histograma
y_i=NULL
for (i in 1:1000){
   y_i=append(y_i,c_i(i))
}

hist(y_i,breaks=50,freq=F,main="Histograma de varianzas")
curve(dchisq(x,df=N-1), from = 850,to=1200,col=2,lwd=4,add=TRUE) # ajustar limites
abline(v=N-1,col=4,lwd=3)# Reemplazar v=9 si n=10
abline(v=999*var(data)/7,col=6,lwd=3)
999*var(data)/7-N+1
```

```
#Vector de medias e histograma de medias
y_i=NULL
for (i in 1:1000){
```

```
    y_i=append(y_i,x_barra(i))
  }
  #Distribucion estandarizacion
  x_i=NULL
  x_est<-function(n){
    for (i in 1:N){
      x_i<-append(x_i,bootstrap[i,n])
    }
    return(sqrt(N)*(mean(x_i)-mean(data))/sqrt(var(data)))
  }
```

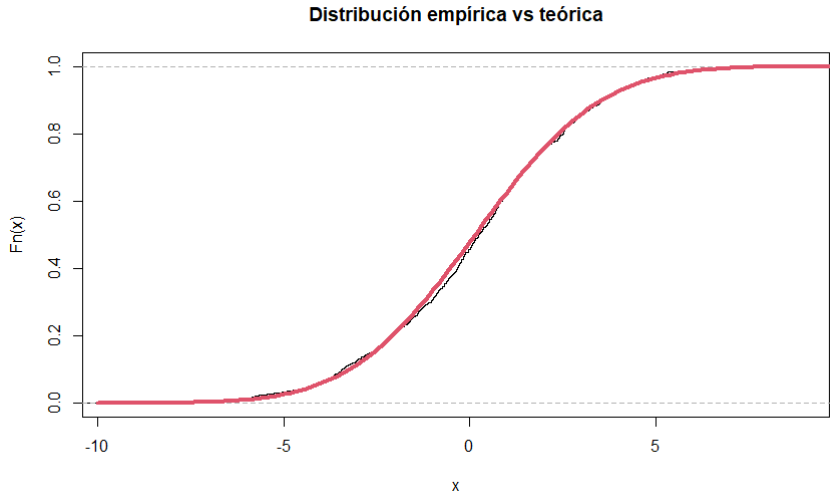


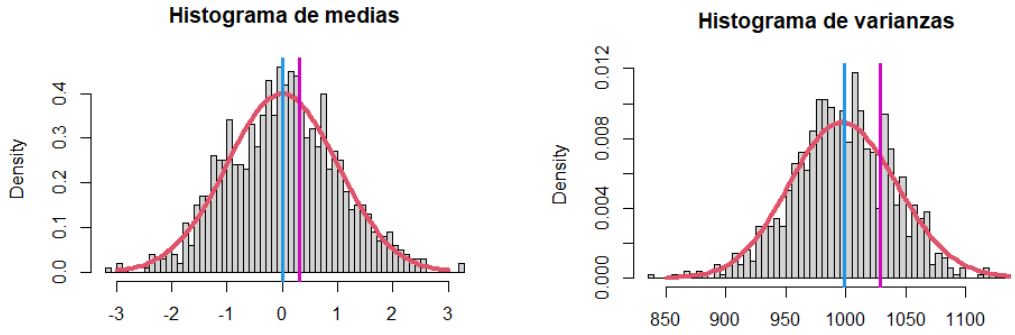


$$\frac{\sqrt{n}(\bar{X}_n-\mu)}{\sigma} \xrightarrow[n\to\infty]{\text{p}} N(\mu, \frac{\sigma^2}{n}) \qquad \frac{s^2(n-1)}{\sigma^2} \sim \chi^2_{n-1}$$

El desajuste entre estos histogramas y las curvas de densidad teóricas pueden verse como una consecuencia del teorema de Glivenko-Cantelli pues como vimos la función de distribución empírica dista bastante de la teórica, por esto mismo las distribuciones asíntoticas usadas para estimar la media y la varianza no son muy buenas. Sin embargo, inspirándonos en el teorema central del límite, si aumentamos el tamaño de muestra obtendremos una mejor aproximación. Esto es precisamente lo que afirma el teorema de Glivenko- Cantelli

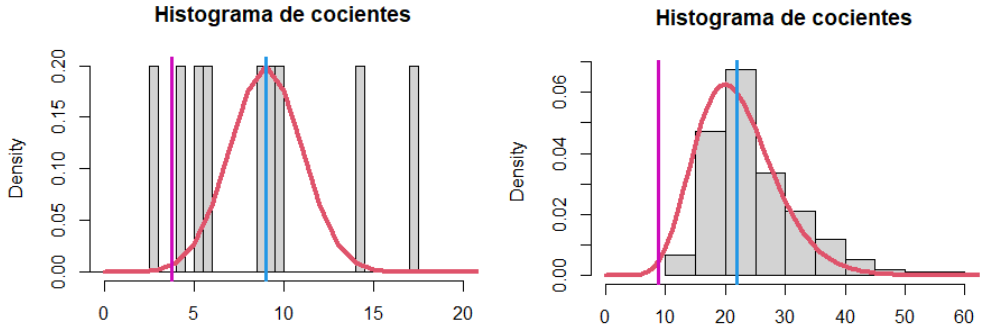
Así, si repetimos este experimento para un tamaño de muestra $n = 1000$.





Ahora bien, este proceso puede ser usado para evaluar otras estadísticas. Sin embargo, conocer la distribución dada por el remuestreo requerirá el uso de otras herramientas. Estudiaremos el caso del coeficiente de variación $\hat{c} = \frac{s}{\bar{x}}$.Generaremos la muestra bootstrap a partir de la siguiente rutina:

```r
#Cociente de variacion
x_i=NULL
err_i<-function(n){
  for (i in 1:N){
    x_i<-append(x_i,s_i(i)/x_barra(i))}
return(x_i)
}
#Vector de cocientes e histograma
y_i=NULL
for (i in 1:1000){
  y_i=append(y_i,err_i(i))
}
hist(y_i,breaks=50,freq=F,xlim = c(0, 60),main="Histograma de cocientes")#Cambiar limites
curve(dnorm(x,mean=9,sd=2), from = 0,to=100,col=2,lwd=4,add=TRUE)# Curva N=10
10-sqrt(var(data))/mean(data)# Sesgo n=10
curve(dchisq(x,df=22), from = 0,to=100,col=2,lwd=4,add=TRUE)#Curva N=1000
22-sqrt(var(data))/mean(data)#Sesgo n=1000
```



$n = 10$ $n = 1000$

$B[\hat{c}] = 1.24$ $B[\hat{c}] = 13.24$

Podemos observar que para este conjunto de datos el modelo $\chi(22)$ resulta ajustarse bastante bien al histograma. La elección del modelo fue hecha simplemente reconociendo la forma del histograma. Sin embargo, para calcular el los grado de libertad se hizo uso de la siguiente rutina que implementa el la función *optim* e R(función de optimización numérica).

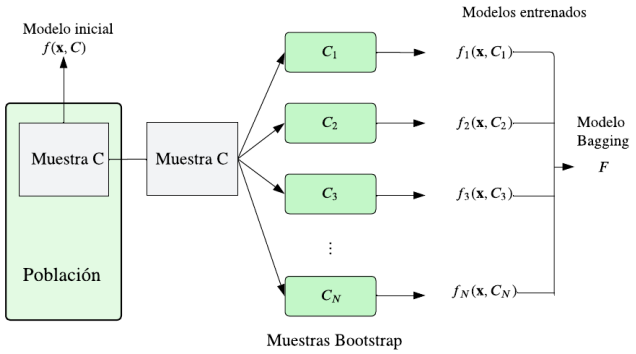
```r
fChi<-function(n){
fc = 1 / (2^(n/2)* gamma(n/2))* x^(n/2-1)* exp(-x/2)
L=-sum(log(fc))
}
solChi=optim(22,fChi);
N=solChi$par[1]
fChi = 1 / (2^(N/2)* gamma(N/2))* pts^(N/2-1)* exp(-pts/2)
lines(pts,fChi,col=4,lwd=2)
```

3.2 Bagging:

Motivación:

El bagging (Agregación al Bootstrap) es una aplicación del bootstrapping propuesta por Leo Breiman en 1994, enfocada a problemas de clasificación estadística y de regresión, esta técnica en la mayoría de los casos produce una disminución en la varianza de los resultados, adicionalmente presenta una disminución del riesgo por sobre entrenamiento de tal forma que al final el modelo resultante se ajuste a los datos de prueba, comúnmente es utilizado para arboles de decisión.

La idea base en la que se fundamenta el método es la siguiente: A partir de un conjunto finito de m datos $D = \{(x_n, y_n), 1 \leq n \leq m\}$, suponemos que existe una relación entre x_n y y_n de tal manera que se obtiene un modelo f que depende de x_n y que predice el valor de y_n, luego si tenemos un conjunto de muestras $\{D_i\}_{1 \leq i \leq N}$ de m datos tal que las muestras son independientes, representativas y que cada una de ellas consta de un modelo f_i que la predice, el objetivo será según la tarea (regresión o clasificación) promediar los resultados de tales modelos para crear un modelo resultante F que tenga una varianza más pequeña o tomar como modelo F el promedio mas alto de voto entre todas las clasificaciones.



Fundamentacion:

Sea $C = \{(x_n, y_n), n = 1, \cdots, m\}$ un conjunto de datos de aprendizaje con una distribución desconocida D, suponga que a dichos datos se puede ajustar(entrenar) un modelo inicial $f(x_n, C) = y_n$ y que adicionalmente tenemos una secuencia de N conjuntos de aprendizaje $\{C_i\}_{i=1,\cdots,N}$ que constan de m datos independientes provenientes de la misma distribución que C, tales que para estos conjuntos esta asociado de igual manera un modelo $f_i(\mathbf{x}, C_i)$, entonces para crear el nuevo modelo F y conseguir que este sea mejor que los propuestos anteriormente, se trabaja con la secuencia de modelos generados $\{f_i(\mathbf{x}, C_i)\}_{i=1,\cdots,N}$ de donde surgirá un modelo mucho mejor que el modelo inicial $f(x_n, C)$.[3]

Para ello, si el problema es de regresión es natural pensar a F como promedio de todos los modelos pertenecientes a la secuencia anteriormente mencionada, es decir:

$$F = \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x}, C_i)$$

Ahora, si se están manejando modelos para clasificación, podemos tomar al nuevo modelo como:

$$F = sign\left(\frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x}, C_i)\right) \text{ o } F = argmax_l \sum_{t \in \{f_i\}} (T_t = l)$$

Una limitación para dicho proceso de mejoramiento es que normalmente no se suelen tener tantas observaciones o datos, de hecho en algunos casos únicamente se tiene el conjunto inicial de observaciones C, luego para obtener la cantidad necesaria de conjuntos se sigue el siguiente algoritmo estrechamente relacionado con el bootstraping[4]:

Algoritmo:

1. Se construye una muestra bootstrap $C_i = \{(x_n^{(i)}, y_n^{(i)}), 1 \leq n \leq m\}$ con reemplazo y que siga la misma distribución que el conjunto de aprendizaje C.
2. Se calcula un estimador o modelo bootstraping (se entrena el modelo) $f_i(\mathbf{x}, C_i)$.
3. Se realizan los dos pasos anteriores según la cantidad N de modelos que se quieran promediar.
4. El nuevo modelo o estimador resultante del proceso de Bagging es $F = \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}, C_i)$ o $F = sign\left(\frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}, C_i)\right)$ según el caso.

Es decir, entrenamos cada modelo individualmente y creamos un nuevo modelo combinando por medio del promedio. Algo importante para resaltar es que, en la teoría es frecuente definir el estimador resultante del bagging como $E_C[f_i(\mathbf{x}, C_i)]$, sin embargo, dependiendo de la naturaleza del problema a la cual se pretenda enfocar no puede ignorarse que en dicha definición es posible que $N = \infty$, lo que en la practica no es factible, sin embargo, una aproximación para dicho valor generalmente bien aceptado es el dado mediante una simulación con el método de Montecarlo.

Ahora, teniendo en cuenta que el objetivo principal es mejorar el estimador en términos de varianza y sobre ajuste, es importante saber en que casos el procedimiento realizado con el algoritmo de Bagging en efecto produce un modelo mucho mejor que el manejado inicialmente, todo dependerá de la estabilidad con la cual se construye el modelo inicial $f(\mathbf{x}, C)$, pues si al realizar algunos cambios en los datos que se encuentran en C no se generan mayores inconvenientes para este estimador, entonces el nuevo modelo F hallado gracias a los conjuntos creados con remuestreo no se encuentra muy lejos del estimador inicial y por tanto las mejoras no son muy significativas, luego el verdadero objetivo será aplicar dicho método cuando se presenten cambios significativos en el modelo $f(\mathbf{x}, C)$ al realizar cambios en el conjunto de aprendizaje C.

Por lo anterior, cabe aclarar que el método siempre nos da un estimador de los datos pero que en algunos casos es mucho mejor que en otros y por tanto se suele utilizar en problemas específicos, para efectos de lo que se pretende mostrar a continuación tómese $f_i(\mathbf{x}, C_i) = f_i(\mathbf{x})$.

Observe el efecto del Bagging sobre la varianza de un modelo:[2]

Supongamos que en base a cierto problema deseamos estimar un parámetro θ utilizando los modelos f_i, tómese e_i como el error resultante de la estimación realizada por medio del $i-$ésimo modelo, por tanto tomando dicho e_i como una variable aleatoria, podemos suponer que esta se distribuye normalmente $e_i \sim \mathcal{N}(0, \sigma^2)$ y definimos la covarianza de cualesquiera dos errores como $cov(e_i, e_j) = \eta$.

En primer lugar, si $Var[e_i] = \sigma^2$, entonces por la definición de varianza dada previamente, se tiene que

$$Var[e_j] = E[e_j^2] - (E[e_j])^2$$

Pero, se sabe que los errores se distribuyen normalmente con media 0, así $E[e_j] = \mu = 0$, por tanto $Var[e_j] = E[e_j^2]$, de donde $E[e_j^2] = \sigma^2$, por tanto se concluye que la varianza del modelo $j-$ésimo es σ^2.

Por otro lado, si analizamos la covarianza del error e_j y el error e_i con $i \neq j$, tenemos que:

$$\text{Cov}(e_i, e_j) = E(e_j e_i) - E[e_i]E[e_j] = \eta$$

Pero $E[e_i] = E[e_j] = 0$, entonces se concluye que $E[e_i e_j] = \eta$, por tanto $\text{cov}(f_i(\mathbf{x}), f_j(\mathbf{x})) = \eta$. Luego, hallamos la varianza del modelo obtenido F :

$$
\begin{aligned}
Var(F) &= Var\left(\frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x})\right) \\
&= \frac{1}{N^2} Var\left(\sum_{i=1}^{N} f_i(\mathbf{x})\right) \\
&= \frac{1}{N^2}\left(\sum_{i=1}^{N} Var(f_i(\mathbf{x})) + \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} \mathrm{Cov}(f_i(\mathbf{x}), f_j(\mathbf{x}))\right) \\
&= \frac{1}{N^2}\sum_{i=1}^{N} Var(f_i(\mathbf{x})) + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} \mathrm{Cov}(f_i(\mathbf{x}), f_j(\mathbf{x}))
\end{aligned}
$$

Aprovechando los resultados anteriormente mostrados y las propiedades de la varianza y covarianza, se deduce que

$$\sum_{i=1}^{N} Var(f_i(\mathbf{x})) = \sum_{i=1}^{N} \sigma^2 = \sigma^2 \sum_{i=1}^{N} 1 = \frac{1}{N}\sigma^2$$

y adicionalmente

$$\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} \mathrm{Cov}(f_i(\mathbf{x}), f_j(\mathbf{x})) = \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} \eta = \frac{1}{N}(N-1)\eta$$

Entonces la expresión que obtenemos para la varianza del modelo esta dada por:

$$Var(F) = \frac{\sigma^2}{N} + \frac{(N-1)\eta}{N}$$

Esta expresión se puede analizar de una manera que puede resultar de mucho provecho, pues para dos de los modelos entrenados se pueden tener alguna de las siguientes opciones:

- Si los modelos que surgen de las distintas muestras bootstrap están demasiado correlacionados, es decir se parecen, entonces se va a tener que $\sigma^2 \approx \eta$ luego, tomando el limite de la varianza de F cuando η tiende a σ^2:

  $$\lim_{\eta\to\sigma^2} Var(F) = \lim_{\eta\to\sigma^2} \frac{\sigma^2}{N} + \frac{N-1}{N}\eta = \frac{(1+N-1)}{N}\sigma^2 = \sigma^2.$$

  Esto es, el modelo resultante del algoritmo tiene la misma varianza que el modelo inicial cuando los modelos resultantes de las muestras están demasiado correlacionados, por tanto si se quería mejorar un modelo en base a su varianza, el bagging no fue de demasiada utilidad.

- Si los modelos que surgen de las distintas muestras bootstrap tienen una correlación demasiado pequeña o son independientes, es claro que $\eta \approx 0$ entonces al tomar el limite de la varianza de F cuando η tiende a 0 se obtiene:

  $$\lim_{\eta\to 0} Var(F) = \lim_{\eta\to 0} \frac{\sigma^2}{N} + \frac{N-1}{N}\eta = \frac{(1+N-1)}{N}\sigma^2 = \sigma^2 = \frac{\sigma^2}{N}$$

  Por tanto la varianza del modelo final es mucho mejor a la varianza del modelo inicial y de los modelos resultantes por el bootstrap.

De lo anterior concluimos que el proceso de Bagging en algunos casos nos brinda mejoras en cuanto a la varianza resultante y si no lo hace nos otorga la varianza con la que ya se estaba trabajando pero nunca aumenta la varianza, por tanto a lo sumo se obtiene un modelo igual pero no peor que el planteado con anterioridad.

Para ejemplificar lo descrito se realizará el siguiente ejemplo:

Suponga que se quiere crear un modelo en base a datos recopilados sobre la diabetes, el siguiente ejemplo mostrará lo ventajoso que puede ser la utilización de los algoritmos como el bagging:

```
#Librerias necesarias
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from tabulate import tabulate
from sklearn.utils import resample
from matplotlib.image import NonUniformImage
from sklearn.datasets import make_regression
from sklearn.ensemble import BaggingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.utils.sparsefuncs import mean_variance_axis

#Se sube la base de datos
diabetes_x,diabetes_y=datasets.load_diabetes(return_X_y=True)

#Se escogen los datos que se desean entrenar y se definen los vectores:

y=np.array([[i]  for i in diabetes_y])
x=diabetes_x
n=10

error = np.zeros(n)
sesgo = np.zeros(n)
varianza = np.zeros(n)
modelos=[i for i in range(n)]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
escala = StandardScaler()
escala.fit(x_train)
x_traine =escala.transform(x_train)
x_teste = escala.transform(x_test)

modelo = DecisionTreeRegressor(criterion="squared_error")
modelo.fit(x_traine, y_train)
prediccioninicial = modelo.predict(x_teste)
```

Miramos cual es el error cuadrático medio de la predicción inicial, mediante el código

```
mse_modelo= mean_squared_error(y_test, prediccioninicial)
print(mse_modelo)
```

Obteniendo la salida

```
5312.511111111111
```

Creamos 10 modelos distintos entrenados con remuestreo bootstraping para ver si existe alguna mejora:

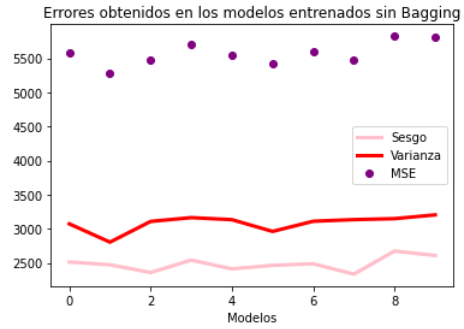
```
#modelos entrenados
modelo1 = DecisionTreeRegressor(criterion="squared_error")
for j in range(1,11):
   y_pred = np.empty((y_test.shape[0], 25))
   #Se realizan 25 remuestreos bootstrap
   for i in range(0,25):
       x_, y_ = resample(x_traine, y_train)
       modelo1.fit(x_, y_)
       y_pred[:, i] = modelo1.predict(x_teste)

   error[j-1] = np.mean( np.mean((y_test - y_pred)**2, axis=1, keepdims=True) )
   sesgo[j-1] = np.mean( (y_test - np.mean(y_pred, axis=1, keepdims=True))**2 )
   varianza[j-1] = np.mean( np.var(y_pred, axis=1, keepdims=True))

#Mostramos los errores respectivos de los modelos
```

```
    tabla= {"Modelo":modelos[1:],"Error": error[1:],"Varianza:": varianza[1:],"Sesgo:":sesgo
    [1:]}
    print(tabulate(tabla,headers="keys",tablefmt="fancy_grid",stralign="left",floatfmt=".16f
    "))
```

Vemos que algunos de estos modelos arrojan mejores varianzas y errores que el modelo inicial, esto se ve mejor representando en la gráfica:

```
    x1=[i for i in range(n)]
    plt.scatter(x1, error, label='MSE',color="purple")
    plt.plot(x1, sesgo, label='Sesgo',color="pink", linewidth=3)
    plt.plot(x1, varianza, label='Varianza',color="red",linewidth=3)
    plt.title('Errores obtenidos en los modelos de entrenamiento sin Bagging',fontname='
    Franklin Gothic Medium', fontsize=12)
    plt.legend(loc="upper right")
    plt.xlabel("Modelos")
    plt.show()
```

Modelo	Error	Varianza:	Sesgo:
1	5277.3688888888891597	2804.2212266666665528	2473.1476622222226069
2	5467.8266666666659148	3108.1755022222223488	2359.6511644444440208
3	5704.3786666666674137	3163.5236266666656775	2540.8550400000003719
4	5546.0693333333329065	3133.8395022222221087	2412.2298311111117073
5	5425.9333333333333940	2961.5945955555548608	2464.3387377777776237
6	5598.2115555555556057	3110.6440533333334315	2487.5675022222221742
7	5470.3315555555554965	3134.4935111111112747	2335.8380444444442219
8	5822.4017777777780793	3148.6706488888889908	2673.7311288888890886
9	5813.0159999999996217	3204.0448711111112061	2608.9711288888888703



Sin embargo, esto no nos es suficiente, pues algunos modelos también presentaron errores más grandes y varianzas muy significativas, luego vamos a crear un modelo utilizando el Bagging:

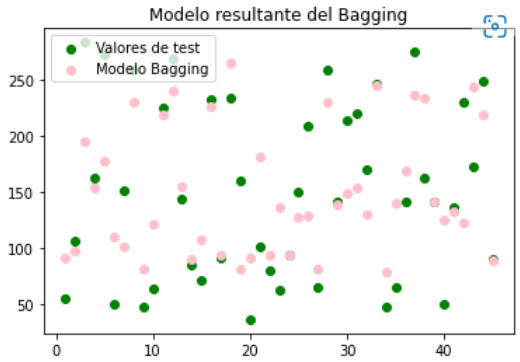
```
    #Usamos Bagging
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
    Modelo_Bagging = BaggingRegressor(base_estimator=DecisionTreeRegressor(),n_estimators
    =25).fit(x_train,y_train.ravel())
    pred_Bagg = Modelo_Bagging.predict(x_test)

    print('MSE para el Modelo con Bagging: {}'.format(mean_squared_error(y_test, pred_Bagg))
    )
```

Obteniendo la salida

```
MSE para el Modelo con Bagging: 2417.5291022222223
```

Se observa que todos los valores resultantes del modelo son más pequeños en comparación a los modelos entrenados con anterioridad, además se puede ver como el Bagging predictor se ajusta mucho mejor a los datos de test

```
plt.scatter(range(1, len(y_test) + 1),y_test, label='Valores de test',color="green")
plt.scatter(range(1, len(y_test) + 1),pred_Bagg, label='Modelo Bagging',color="pink")
plt.legend(loc="upper left")
plt.title("Modelo resultante del Bagging",fontname='Franklin Gothic Medium', fontsize=12)
plt.show()
```



En conclusión, se observa una gran diferencia entre los errores obtenidos en los distintos modelos entrenados únicamente con remuestreo y el error resultante del modelo creado con Bagging, pues el error de este ultimo a pesar de seguir siendo grande es mucho más pequeño que todos los errores obtenidos con anterioridad, teniendo en cuenta que medimos el error MSE, esto refuerza que el Bagging reduce la varianza de una manera significativa.

3.3 Árboles de decisión:

Motivación:

Los arboles de decisión son un algoritmo de aprendizaje de maquina de tipo supervisado, es decir, a partir de ejemplos de pares de inputs y outputs, propone una función que asigne a cada input su correspondiente output. En el caso en el que los outputs son discretos se trata de un árbol de clasificación y en el caso en el que los outputs son continuos se trata de un árbol de regresión.

Los arboles se estructuran de la siguiente forma: Cada nodo interno del árbol representa una decisión acerca de un atributo(elemento) del vector de características, los arcos salientes del nodo representan los posible valores(o rango de valores) que puede tomar dicho atributo y los nodos hoja representan el valor de la variable de salida a ser retornado por el algoritmo.

Algoritmo:

El algoritmo para construir el árbol de decisión consiste en particionar los datos(y en general el espacio de características) de forma recursiva en base a una secuencia de "decisiones". Esto es, se empieza con el nodo raíz que contiene todas las observaciones disponibles, luego se escoge un atributo del vector de características sobre el cual dividir los datos, después se forman nuevos nodos hijos para cada uno de los valores(o rango de valores) que toma el atributo y que contienen las observaciones cuyo valor del atributo coincide con el del nodo, finalmente se repite el proceso sobre cada uno de los nodos hijo hasta alcanzar cierto criterio, comúnmente es que el numero de observaciones restantes sea 1.

Nuestro objetivo es entonces, encontrar un árbol que sea consistente con nuestros datos y que además sea lo más pequeño posible, no obstante resulta inviable buscar de forma eficiente entre 2^{2^n} arboles posibles[8]. Sin embargo, existen heurísticas para construir un árbol que se aproxime al más pequeño posible. En una en particular, la idea es que en el algoritmo del árbol de decisión, se realicen las divisiones sobre los atributos más "importantes" primero, más importante refiriéndose a aquel atributo que proporciona una división en términos de la clasificación de las observaciones lo mas marcada posible, esto es, que los datos que tengan la misma clasificación queden en el mismo grupo.

Una forma de medir la "importancia" de un atributo es mediante el concepto de *entropía* proveniente del área de la *teoría de la información*. La entropía de una variable aleatoria representa la cantidad de "incertidumbre" o "información" que se tiene sobre los resultados de la variable. Por ejemplo, el lanzamiento de una moneda común tiene 1(bit) de entropía, mientras que el de una moneda cargada donde con una probabilidad de 0.99 es cara, tiene una entropía cercana a 0.[8]

Entonces para medir la calidad de dividir sobre un atributo queremos medir la diferencia entre la entropía "antes" y la entropía "después" de hacer la división de las observaciones sobre el atributo, esto es lo que llamaremos ganancia de información, que es precisamente lo que buscamos con decir la importancia de un atributo, luego vamos a escoger el atributo que nos proporcione la mayor ganancia de información.

Formalmente. la entropía de una variable aleatoria V que toma los valores v_k, se define como[8]:

$$H(V) = -\sum_k P(v_k)\log_2(v_k)$$

Para nuestro problema de clasificación, en el que el valor de salida puede tomar los valores $0, 1, \cdots, k-1$, definimos a la proporción de observaciones cuyo atributo A toma el valor k, en el nodo m, donde n_m es el número de observaciones como[9]:

$$p_{mk} = \frac{1}{n_m}\sum_{y \in Q_m} I(A = k)$$

Entonces la entropía del conjunto de observaciones Q_m del nodo m se define como[9]:

$$H(Q_m) = -\sum_k p_{mk}\log_2 p_{mk}$$

Ahora bien, el problema de encontrar el atributo mas importante es ligeramente distinto dependiendo si los atributos toman un número finito de valores o no. Para el primer caso, tenemos lo siguiente:

Sean las n_m observaciones en el nodo m representadas por Q_m. Ahora bien para un atributo A que toma r valores distintos, se divide el conjunto de observaciones Q_m del nodo m en $Q_{m_1}, Q_{m_2}, \cdots, Q_{m_r}$ subconjuntos cada uno de tamaño $n_{m_1}, n_{m_2}, \cdots, n_{m_r}$ y con proporciones $p_{m_i 0}, p_{m_i 1}, \cdots, p_{m_i k-1}$ para $1 \leq i \leq r$. Luego para cada valor de A, necesitaremos $H(Q_{m_i})$ bits de información para poder decidir sobre el atributo de salida. Como la probabilidad de escoger una observación de Q_m que tenga el valor k en el atributo A es $\frac{n_{m_k}}{n_m}$, definimos la entropía esperada después de hacer la división de Q_n sobre el atributo A como:

$$\sum_i \frac{n_{m_i}}{n_m} H(Q_{m_i})$$

Dicho esto, definimos la ganancia de dividir sobre el atributo A en el nodo Q_m como la reducción esperada en la entropía:

$$G(A) = H(Q_m) - \sum_i \frac{n_{m_i}}{n_m} H(Q_{m_i})$$

Por lo tanto, en cada paso del algoritmo del árbol de decisión, escogemos el atributo que maximice la ganancia:

$$A^* = \text{argmax}_A G(A)$$

Por otro lado para el caso en el que los atributos toman un número infinito de valores, no podemos generar un numero infinito de ramificaciones, en vez de eso escogemos un atributo, un punto de corte y particionamos el conjunto de observaciones en dos: los que están por encima del punto de corte y los que están por debajo del punto de corte.

Al igual que el caso anterior, sean las n_m observaciones en el nodo m representadas por Q_m. Para cada posible corte $\theta = (j, t_m)$ en la que se divide sobre el atributo j y con el punto de corte t_m, se dividen los datos en $Q_m^{left}(\theta)$ y $Q_m^{right}(\theta)$, siendo:

$$Q_m^{left}(\theta) = \{(x, y)|x_j \leq t_m\}$$

$$Q_m^{right}(\theta) = Q_m / Q_m^{left}(\theta)$$

Por lo tanto, la calidad de la división en el nodo m se calcula como:

$$G(Q_m, \theta) = H(Q_m) - \left(\frac{n_m^{left}}{n_m} H(Q_m^{left}) + \frac{n_m^{right}}{n_m} H(Q_m^{right}) \right)$$

Luego se selecciona el parámetro que maximiza la ganancia:

$$\theta^* = \text{argmax}_\theta G(Q_m, \theta)$$

Sobreajuste:

Por otra parte, un problema importante de los arboles de decisión y en general de los modelos de aprendizaje supervisado es el sobreajuste a los datos de entrenamiento, este fenómeno consiste en que el modelo "memoriza" características irrelevantes de los datos de entrenamiento de forma que el modelo pierde la capacidad de encontrar patrones que sean generalizables a todos los datos y por lo tanto tendrá un desempeño muy pobre con datos que no haya visto antes.

La primera forma de contrarrestar el sobreajuste es añadiendo mas datos al conjunto de entrenamiento, de esta forma es menos probable que los datos de entrenamiento evidencien características irrelevantes y que por lo tanto no sean generalizables a todo el espacio de características. Sin embargo, esto no siempre es posible, por lo que la segunda opción es hacer una técnica llamada *decision tree pruning*, en la cual se eliminan los nodos del árbol que dividan las observaciones sobre atributos irrelevantes. Un indicador de la irrelevancia de un atributo es la ganancia de información, ya que si un atributo es irrelevante es esperable que este divida las observaciones en conjuntos cuyas proporciones p_{km} de clases sean aproximadamente las mismas que las del conjunto antes de ser dividido, es decir, que esta división tendría una ganancia de información cercana a 0.[8]

La siguiente pregunta natural es, que tan grande debe ser la ganancia de información para dividir sobre un atributo, para responder esta pregunta podemos implementar un test de significancia estadística. Primero suponemos que no hay ningún patrón en los datos, esta es la hipótesis nula. Luego bajo esta suposición, calculamos cuanto se desvían los datos sobre la ausencia de patrón alguno. Si el grado de desviación es estadísticamente significativo(normalmente si tiene una probabilidad de 0.05 o menos), entonces se considera que existe evidencia significativa de un patrón en los datos y se rechaza la hipótesis nula.

Suponga que queremos realizar el test de significancia estadística sobre un atributo A que toma r valores distintos, y que divide el conjunto de observaciones Q_m del nodo m en $Q_{m_1}, Q_{m_2}, \cdots, Q_{m_r}$ subconjuntos cada uno de tamaño $n_{m_1}, n_{m_2}, \cdots, n_{m_r}$ y con proporciones $p_{m_i 0}, p_{m_i 1}, \cdots, p_{m_i k-1}$ para $1 \leq i \leq r$. Por la hipótesis nula, supongamos que el atributo A es irrelevante, entonces para medir la desviación podemos comparar las proporciones de las clases $p_{m_i k}$ del cada subconjunto Q_{m_i} con las proporciones esperadas $\hat{p}_{m_i k}$ asumiendo que no hay ningún patrón, donde:

$$\hat{p}_{m_i k} = p_{m_i k} \cdot \frac{n_{m_i}}{n_m}$$

Entonces podemos calcular la desviación total como[8]:

$$\Delta = \sum_r \sum_k \frac{(p_{m_r k} - \hat{p}_{m_r k})^2}{\hat{p}_{m_r k}}$$

Bajo la hipótesis nula, Δ esta distribuido bajo la distribución χ^2, entonces mediante un software estadístico podremos confirmar o rechazar la hipótesis nula y consecuentemente decidir si remover el nodo del árbol.

4. Aplicación en el Machine Learning

```
#Librerias Necesarias

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, mean_squared_error, mean_absolute_error
```

El dataset viene incluido en la librería de Machine Learning Scikit-learn, entonces lo podemos cargar mediante la siguiente función:

```
#Base de datos
house_prices = fetch_california_housing(as_frame=True)
```

Se cargan los datos y se visualizan las 5 primeras filas.

```
X = house_prices.data
y = house_prices.target
pd.concat([X, y], axis=1).head()
```

Contamos con 20640 datos en total:

```
print(X.shape, y.shape)

(20640, 8) (20640,)
```

Se convierten los datos en arreglos de Numpy:

```
X = X.values
y = y.values
```

A continuación, se dividen los datos en conjuntos de entrenamiento y testeo, después vamos a declarar un modelo de árbol de decisión y otro modelo de Bagging con arboles de decisión. Luego entrenamos los modelos con los datos de entrenamiento y mostramos los resultados en los datos de testeo.

```
# Dividimos los datos en conjuntos de entrenamiento y testeo
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state=42)
# Arbol de decision
reg = DecisionTreeRegressor().fit(X_train, y_train)
# Bagging con arboles de decision
bagg = BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_estimators=10,
  random_state=0).fit(X_train, y_train)
# Realizamos las predicciones en los datos de testeo
y_pred = reg.predict(X_test)
y_pred_bagg = bagg.predict(X_test)
```

```
print(f"No bagging MSE: {mean_squared_error(y_pred, y_test)}")
print(f"No bagging MAE: {mean_absolute_error(y_pred, y_test)}")

print(f"Bagging MSE: {mean_squared_error(y_pred_bagg, y_test)}")
print(f"Bagging MAE: {mean_absolute_error(y_pred_bagg, y_test)}")
```

Como salida obtenemos

```
No bagging MSE: 0.48904179333768105
No bagging MAE: 0.4541115458937198
Bagging MSE: 0.21837095896189854
Bagging MAE: 0.33232928019323676
```

Miramos cuales son las varianzas obtenidas para los modelos:

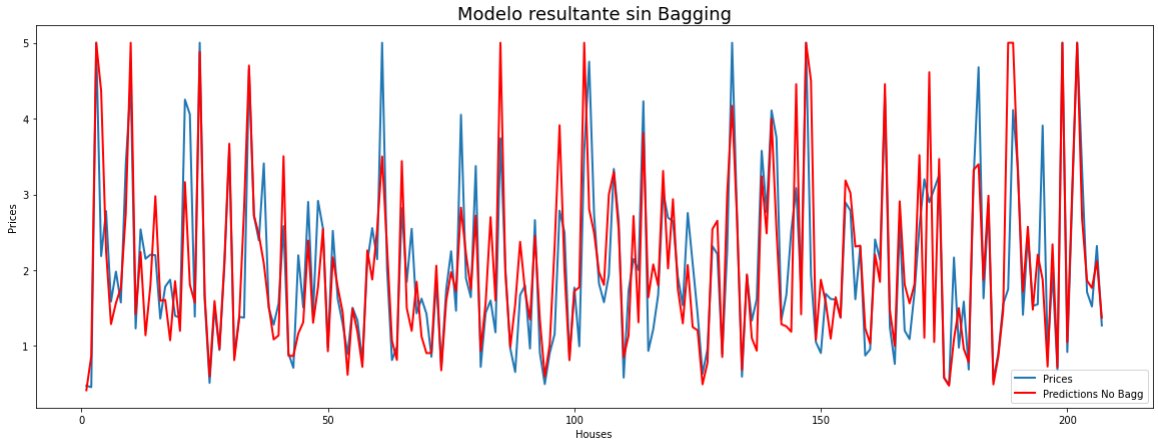
```
# Varianza de las predicciones
print(f"La varianza de las predicciones usando bagging es {np.var(y_pred_bagg)}") # bagging
print(f"La varianza de las predicciones sin usar bagging es {np.var(y_pred)}") # no bagging
```

Como salida obtenemos

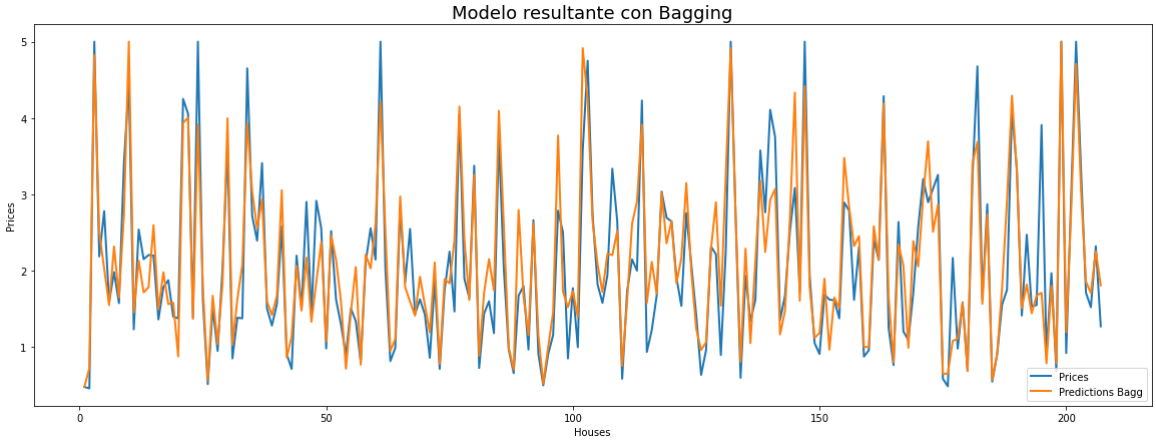
```
La varianza de las predicciones usando bagging es 1.0600248155396956
La varianza de las predicciones sin usar bagging es 1.2898278105794583
```

Ahora visualizamos los resultados de ambos modelos:

```
plt.figure(figsize=(20,7))

plt.plot(range(1, len(y_test) + 1), y_test,label='Prices',linewidth=2)
plt.plot(range(1, len(y_test) + 1), y_pred,label='Predictions No Bagg',color='red', linewidth=2)
plt.xlabel('Houses')
plt.ylabel('Prices')
plt.title('Modelo resultante sin Bagging',fontname='Franklin Gothic Medium', fontsize=18)
plt.legend()
plt.show()
```




```
plt.figure(figsize=(20,7))

plt.plot(range(1, len(y_test) + 1), y_test,label='Prices',linewidth=2)
#plt.plot(range(1, len(y_test) + 1), y_pred,label='Predictions No Bagg')
plt.plot(range(1, len(y_test) + 1), y_pred_bagg,label='Predictions Bagg',linewidth=2)
plt.xlabel('Houses')
plt.ylabel('Prices')
plt.title('Modelo resultante con Bagging',fontname='Franklin Gothic Medium', fontsize=18)
plt.legend()
plt.show()
```



Sin duda se observa que el modelo creado con bagging sigue de una mejor manera los precios que el modelo creado sin bagging.

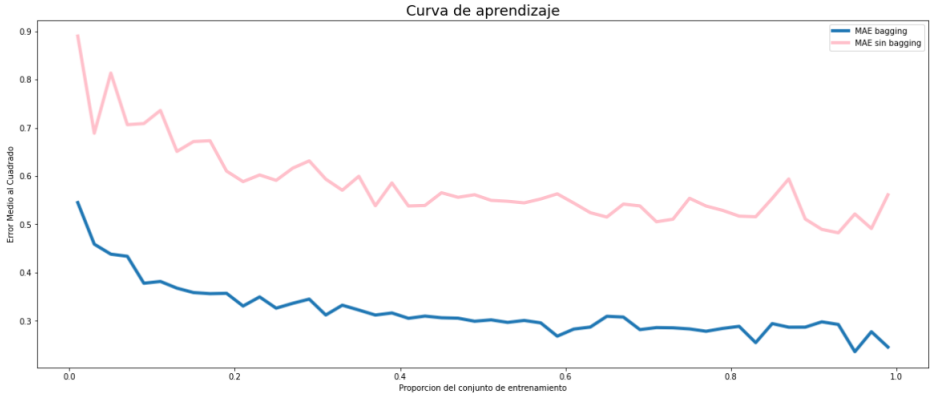
Adicionalmente, queremos saber que tan bueno es el estimador o modelo creado, por tanto utilizamos el siguiente código:

```
# Que tan bueno es el estimador para los datos de prueba.

bagg.score(X_test, y_test)
```

La salida es de 0.8239353198192659, este número indica el coeficiente de determinación de la precisión R^2, de manera más concreta R^2 se define como $\left(1 - \frac{v}{u}\right)$ donde v es la suma residual de cuadrados $\sum(\mathrm{y}_i - \mathrm{y\_pred})^2$, siendo y_i los valores verdaderos del dataset y u la suma del total de cuadrados $\sum(\mathrm{y}_i - \frac{1}{m}\sum \mathrm{y}_i)^2$, como se puede intuir el puntaje máximo que puede obtener un modelo es 1.

Ahora, vamos a graficar la curva de aprendizaje de ambos modelos, en la que medimos el desempeño del modelo en relación a la proporción de los datos con los que fue entrenado.

```
acc_bagg = []
n = 50
split_range = np.linspace(0.01, 0.99, n)

for split in split_range:
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1-split)
  bagg = BaggingRegressor(base_estimator=DecisionTreeRegressor()).fit(X_train, y_train)
  y_pred_bagg = bagg.predict(X_test)
  acc_bagg.append(mean_squared_error(y_pred_bagg, y_test))

acc_no_bagg = []
for split in split_range:
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1-split)
  reg = DecisionTreeRegressor().fit(X_train, y_train).fit(X_train, y_train)
  y_pred_no_bagg = reg.predict(X_test)
  acc_no_bagg.append(mean_squared_error(y_pred_no_bagg, y_test))

plt.figure(figsize=(20, 8))
plt.plot(split_range, acc_bagg, label="MAE bagging",linewidth=4)
plt.plot(split_range, acc_no_bagg, label="MAE sin bagging",color='pink',linewidth=4)
plt.title("Curva de aprendizaje",fontname='Franklin Gothic Medium', fontsize=18)
plt.xlabel("Proporcion del conjunto de entrenamiento")
plt.ylabel("Error Medio al Cuadrado")
plt.legend()
plt.show()
```



Por último, se mostrará como mejoran los errores de los modelos según el número de muestras bootstrap que se tomen, es decir, se observará si los modelos mejoran teniendo un número mayor de remuestreo.

```
MSE_pred=np.zeros(60)
MAE_pred=np.zeros(60)
MSE_pred_bagg=np.zeros(60)
MAE_pred_bagg=np.zeros(60)

for i in range(10,70):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.01, random_state
  =42)
    reg = DecisionTreeRegressor().fit(X_train, y_train)
    bagg = BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_estimators=i,
  random_state=0).fit(X_train, y_train)
    y_pred = reg.predict(X_test)
    MSE_pred[i-10]=mean_squared_error(y_pred, y_test)
    MAE_pred[i-10]=mean_absolute_error(y_pred, y_test)
    y_pred_bagg = bagg.predict(X_test)
    MSE_pred_bagg[i-10]=mean_squared_error(y_pred_bagg, y_test)
    MAE_pred_bagg[i-10]=mean_absolute_error(y_pred_bagg, y_test)

plt.figure(figsize=(20,7))

plt.plot(range(10,70),MSE_pred,linewidth=4, color='plum')
plt.plot(range(10,70),MSE_pred_bagg,linewidth=4,color='lightskyblue')
plt.plot(range(10,70),MAE_pred,linewidth=4,color='red')
plt.plot(range(10,70),MAE_pred_bagg,linewidth=4,color='palegreen')
plt.xlabel('Tamano de Muestra')
plt.ylabel('MSE y MAE')
plt.title('Errores obtenidos con Bagging y sin Bagging',fontname='Franklin Gothic Medium',
   fontsize=18)
plt.legend(["MSE Sin Bagging","MSE Con Bagging","MAE Sin Bagging","MAE Con Bagging"],loc="
  upper right")
plt.show()
```

Los resultados, son:

Si se realizan varios experimentos jugando con el número de muestras bootstrap que se utilizaran a la hora de realizar el modelo de Bagging se puede notar que en algún momento el error se estabiliza, por tanto lo

recomendado es no realizar tantas muestras pues a pesar de que en efecto se mejora la precisión del modelo, dado que la varianza se estabiliza dicha precisión no cambia demasiado de un número de muestras N grande a un número de muestras M aún más grande, pero computacionalmente es mucho mejor manejar un numero pequeño de muestras debido al tiempo de espera.

Observación: Para clasificación se suelen usar 50 muestras bootstrap y para regresión 75, en esta implementación se usaron 10 muestras y sin embargo la duración de compilado de la ultima gráfica es de $6 - 10$ minutos.

5. Conclusiones

Como mencionamos anteriormente los métodos de Montecarlo son una herramienta muy poderosa para realizar simulaciones de expresiones matemáticas a las cuales queremos aproximarnos. Entre esta familia de métodos, los derivados de bootstrapping demuestran tener el potencial para ser usados tanto en el contexto de la inferencia estadística como en algunas técnicas de Machine Learning.

La filosofía detrás del método bootstrapping es relativamente sencilla e intuitiva; esto gracias al uso del teorema de Glivenco-Cantelli, este nos dice que con un tamaño de muestra grande la distribución de toda variable aleatoria será muy similar a la suma de funciones indicadoras, adicionalmente al uso dado en este articulo, esta técnica nos permitirá estimar parámetros de una población y simular el comportamiento de muchas estadísticas. En algunos de estos casos, algoritmos de optimización deben ser usados, aunque conozcamos que distribución siguen nuestros conjuntos de datos.

A partir de un conjunto de datos de entrenamiento, bagging promedia a todos los modelos construidos con remuestreo, de manera que se obtiene un modelo para el cual se mejora el estimador en términos de la varianza, es decir, como se evidencio el Bagging mejora la precisión y robustez de un modelo, pero dado que se realiza con técnicas de remuestreo el coste computacional es demasiado elevado. Por tanto es de utilidad saber en que casos es de provecho aplicar tal algoritmo, por ejemplo, como se describió este no resulta muy útil cuando los modelos resultantes de las simulaciones Bootstrapping están muy correlacionados, caso contrario si la correlación es baja. En cualquier caso, algo importante que resaltar y que refuerza la robustez del bagging es que nunca produce un modelo con más varianza que el original.

Finalmente, como la mejora se presenta en términos de precisión, este método es usado principalmente en problemas de clasificación y regresión en los cuales se suele tener una varianza demasiada alta que se quisiera disminuir. En conclusión el bagging es un algoritmo relativamente fácil para mejorar un predictor en términos de otros que ya se han obtenido.

References

[1] J. Humberto Mayorga A. *Inferencia Estadística*. Universidad Nacional de Colombia, 2004.

[2] Leo Breiman. *Bagging Predictors*. URL: https://link.springer.com/content/pdf/10.1007/BF00058655.pdf.

[3] Peter Buhlmann. *Bagging, Boosting and Ensemble Methods*. URL: https://www.researchgate.net/publication/45130375_Bagging_Boosting_and_Ensemble_Methods.

[4] Peter Buhlmann and Bin Yu. *ANALYZING BAGGING*. URL: https://www.researchgate.net/publication/38348647_Analyzing_Bagging.

[5] Liliana Blanco Castaneda. *Probabilidad*. Universidad Nacional de Colombia, 2004.

[6] David V. Hinkleyt. *Bootstrap Method*. URL: https://www.math.wustl.edu/~kuffner/AlastairYoung/HinkleyDiCiccioRomano1988discus pdf.

[7] Daniel Raban. *The Glivenko-Cantelli Theorem and Introduction to VC Dimension*. URL: https://pillowmath.github.io/Expository%20$%5C%$%2020Notes/VC-Dimension-and-Glivenko-Cantelli-Notes.pdf.

[8] Stuart J. Russell. "Artificial Intelligence: A Modern Approach". In: Prentice Hall, 2010. Chap. 18.3. ISBN: 0-13-461099-7.

[9] Scikit-learn. *1.10. Decision Trees*. URL: https://scikit-learn.org/stable/modules/tree.html.

[10] Scikit-learn. *1.11. Ensemble methods*. URL: https://scikit-learn.org/stable/modules/ensemble.html.

[11] Sergio Gonzaleza Salvador Garcia Javier Del Ser and Lior Rokach andFrancisco Herrera. *A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities*. URL: https://www.sciencedirect.com/science/article/abs/pii/S1566253520303195.