Contenido

[Aprendizaje supervisado 1](#_Toc508721371)

[Sobreajuste 1](#_Toc508721372)

[Sub-ajuste 2](#_Toc508721373)

[Arboles de decisión 2](#_Toc508721374)

[Desarrollo 2](#_Toc508721375)

[Entrenamiento y test 4](#_Toc508721376)

[Estimacion del error de generalización – Validacion Cruzada 7](#_Toc508721377)

[Despliegue del modelo: 7](#_Toc508721378)

[Random Forest 8](#_Toc508721379)

[Ventajas 9](#_Toc508721380)

[Desventajas 9](#_Toc508721381)

[Técnica de bagging 9](#_Toc508721382)

[Bagging para mejorar un modelo predictivo 9](#_Toc508721383)

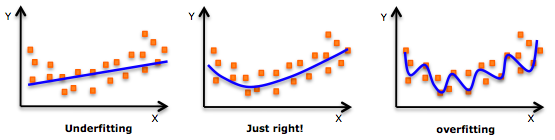
[La técnica de Bagging sigue estos pasos: 9](#_Toc508721384)

[Desarrollo 10](#_Toc508721385)

# Aprendizaje supervisado

Sobreajuste  
  
El sobreajuste significa que el modelo que entrenamos se entrenó "demasiado bien" y ahora, bueno, se ajusta demasiado al conjunto de datos de entrenamiento. Esto generalmente ocurre cuando el modelo es demasiado complejo (es decir, demasiadas características / variables en comparación con el número de observaciones). Este modelo será muy preciso en los datos de entrenamiento, pero probablemente no será muy preciso en datos nuevos o no entrenados. Es porque este modelo no está generalizado, lo que significa que puede generalizar los resultados y no puede hacer ninguna inferencia sobre otros datos, que es, en última instancia, lo que está tratando de hacer. Básicamente, cuando esto sucede, el modelo aprende o describe el "ruido" en los datos de entrenamiento en lugar de las relaciones reales entre las variables en los datos. Obviamente, este ruido no forma parte de ningún conjunto de datos nuevo y no se puede aplicar a él.

Sub-ajuste  
  
A diferencia del sobreajuste, cuando un modelo no está bien ajustado, significa que el modelo no se ajusta a los datos de entrenamiento y, por lo tanto, pasa por alto las tendencias en los datos. También significa que el modelo no se puede generalizar a nuevos datos. Como probablemente haya adivinado (¡o descubierto!), Este suele ser el resultado de un modelo muy simple (no hay suficientes predictores / variables independientes). También podría suceder cuando, por ejemplo, ajustamos un modelo lineal (como la regresión lineal) a datos que no son lineales. Valga la pena decir que este modelo tendrá poca capacidad predictiva (en los datos de entrenamiento y no se puede generalizar a otros datos).



# Arboles de decisión

## Desarrollo

Importacion de librerías indispensables para el taller:

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

import pandas as pd

# importamos la libreria numpy

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import label\_binarize

from sklearn import tree

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.metrics import roc\_curve, auc, roc\_auc\_score

from sklearn.externals import joblib

import matplotlib.pyplot as plt

import graphviz as gv

# seteamos el objeto random a 0

np.random.seed(0)

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

Para la instalacion de graphviz desde la consola de linux:

sudo apt-get install graphviz

luego la instalacion del modulo python correspondiente:

pip3 install graphviz

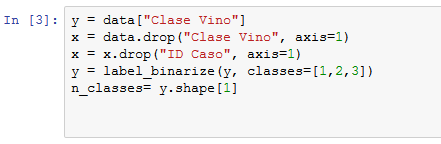
Lectura del archivo Clasificacion



Sea x el conjunto de las variables explicativas (aquellas distintas de la clase)

Sea y , el dataframe con la variable target (Clase Vino)

**Por qué eliminamos la variable ID Caso?**



**Según las tareas que se llevan a cabo en un proceso de aprendizaje supervisado, qué hace la instrucción siguiente?**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y,

test\_size=VALOR,

random\_state=0,

#stratify=y

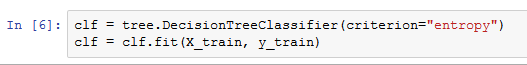
)

Donde VALOR = porcentaje del conjunto de datos usado para test. Defina un valor que no lleve al subajuste o al sobreajuste.

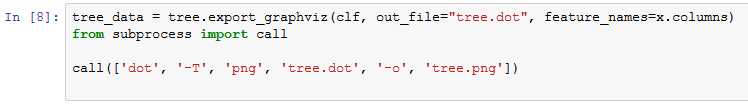
## Entrenamiento y test

Ahora creamos un modelo de Arbol de decisión, basado en el concepto de ganancia de información.

La función fit, ajusta el modelo a los datos de entrenamiento

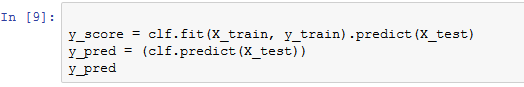


Si se desea ver de manera grafica el árbol de decisión generado, una opcion es generar un archivo de imagen, el cual quedará almacenado en el mismo directorio del archivo ipynb. (La explicación de la instalación del graphviz, se encuentra al comienzo de este documento..)

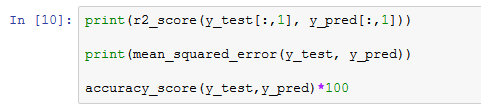


Agregue acá la imagen del árbol de decisión generada. Puede escribir alguna regla de clasificación apartir de este?

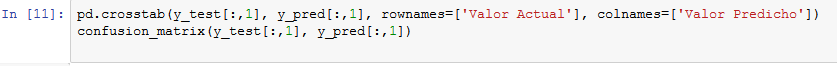
Ahora que se ha generado el modelo, se medirá su score, y se realizará la predicción sobre el conjunto de datos para test.



Que significan las siguientes métricas que se generan a continuación?



La siguiente es la matriz de confusión para la clase 1. Qué puede decir del modelo, con base en ella? Generela también para las clases 2 y 3 y añada sus comentarios del modelo.



Ahora agregue el siguiente fragmento de codigo para calcular el área ROC (área bajo la curva), Por cierto, que es una curva ROC? Que nos dice del modelo??

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:, i], y\_score[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Cálculo de la curva ROC y el area ROC

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test.ravel(), y\_score.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

roc\_auc[0]

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

Ahora puede grafica la curva ROC, incluyendo el siguiente codigo::

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

plt.figure()

lw=2

plt.plot(fpr[2],tpr[2],color="red",

lw=lw, label='Curva ROC(area = %0.2f)' % roc\_auc[0])

plt.plot([0,1],[0,1], color='navy', lw=lw, linestyle='--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.05])

plt.xlabel('Tasa de falsos positivos')

plt.ylabel('Tasa de verdaders positivos')

plt.title('grafico Receiver Operating Characteristic ')

plt.legend(loc='lower right')

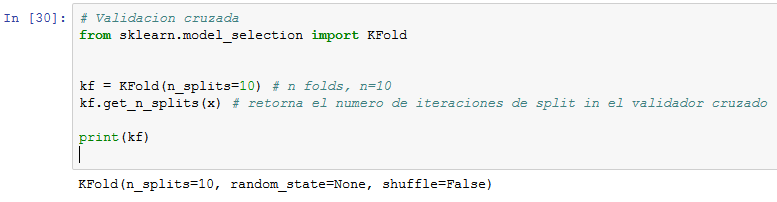
plt.show()

|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

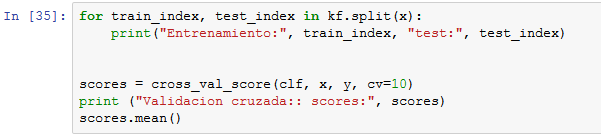
Muestre la curva ROC, para la segunda y tercer clase

## Estimacion del error de generalización – Validacion Cruzada

Se ejecutará la técnica de cross validation con n=1, para garantizar que los resultados son independientes de la partición entre datos de entrenamiento y prueba



Explique qué hace la siguiente instrucción::



**Actividad:** Genere el modelo utilizando el criterio gini, y luego de comparar ambos métodos, indique sus observaciones.(puede incluir tablas comparativas si es el caso)

## Despliegue del modelo:

Una vez se ha validado que el modelo generado tiene una grado de eficacia aceptable, si se desea desplegar para nuevos ejemplos, basta con guardar el modelo a través del modulo joblib y la función dump

Cuando se desea utilizar, se carga con la función load, indicando el modelo a abrir, y se aplica en el conjunto que se desea.





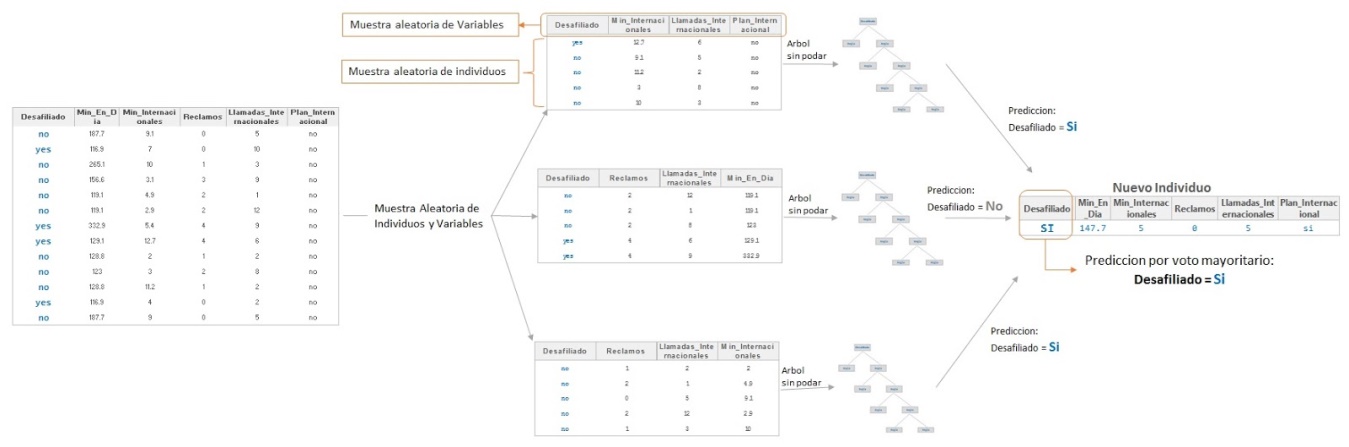


# Random Forest

Random Forest es un algoritmo predictivo que usa la técnica de Bagging para combinar diferentes árboles, donde cada árbol es construido con observaciones y variables aleatorias. El algoritmo construye una larga colección de árboles no correlacionados y luego los promedia

* En forma resumida sigue este proceso:

1. Selecciona individuos al azar (usando muestreo con reemplazo) para crear diferentes set de datos.
2. Crea un árbol de decisión con cada set de datos, obteniendo diferentes árboles, ya que cada set contiene diferentes individuos y diferentes variables en cada nodo.
3. Al crear los árboles se eligen variables al azar en cada nodo del árbol, dejando crecer el árbol en profundidad (es decir, sin podar).
4. Predice los nuevos datos usando el "voto mayoritario", donde clasificará como "positivo" si la mayoría de los arboles predicen la observación como positiva.

<http://2.bp.blogspot.com/-ngn9mrb4mBI/VHVHtVYNK1I/AAAAAAAABPw/39oUK7VBJf4/s1600/RandonForest_2.jpg>

## Ventajas

* Es uno de los algoritmos de aprendizaje más certeros que hay disponible. Para un set de datos lo suficientemente grande produce un clasificador muy preciso.
* Corre eficientemente en bases de datos grandes.
* Puede manejar cientos de variables de entrada sin excluir ninguna.
* Da estimados de qué variables son importantes en la clasificación.
* Tiene un método eficaz para estimar datos perdidos y mantener la exactitud cuando una gran proporción de los datos está perdida.
* Computa los prototipos que dan información sobre la relación entre las variables y la clasificación.
* Computa las proximidades entre los pares de casos que pueden usarse en los grupos, localizando valores atípicos, o (ascendiendo) dando vistas interesantes de los datos.
* Ofrece un método experimental para detectar las interacciones de las variables.

## Desventajas

* Se ha observado que Random forests sobreajusta en ciertos grupos de datos con tareas de clasificación/regresión ruidosas.
* A diferencia de los árboles de decisión, la clasificación hecha por random forests es difícil de interpretar por el hombre.​
* Para los datos que incluyen variables categóricas con diferente número de niveles, el random forests se parcializa a favor de esos atributos con más niveles. Por consiguiente, la posición que marca la variable no es fiable para este tipo de datos.
* Si los datos contienen grupos de atributos correlacionados con similar relevancia para el rendimiento, entonces los grupos más pequeños están favorecidos sobre los grupos más grandes.

## Técnica de bagging

## Bagging para mejorar un modelo predictivo[[1]](#footnote-1)

Una forma de mejorar un modelo predictivo es usando la técnica creada por Leo Breiman que denominó Bagging (o Bootstrap Aggregating). Esta técnica consiste en crear diferentes modelos usando muestras aleatorias con reemplazo y luego combinar o ensamblar los resultados.

### La técnica de Bagging sigue estos pasos:

1. Divide el set de Entrenamiento en distintos sub set de datos, obteniendo como resultado diferentes muestras aleatorias con las siguientes características:

* Muestra uniforme (misma cantidad de individuos en cada set)
* Muestras con reemplazo (los individuos pueden repetirse en el mismo set de datos).
* El tamaño de la muestra es igual al tamaño del set de entrenamiento, pero no contiene a todos los individuos ya que algunos se repiten.
* Si se usan muestras sin reemplazo, suele elegirse el 50% de los datos como tamaño de muestra

1. Luego se crea un modelo predictivo con cada set, obteniendo modelos diferentes
2. Luego se construye o ensambla un único modelo predictivo, que es el promedio de todos los modelos.

### Características de Bagging:

* Disminuye la varianza de un data set al realizar remuestreo con reemplazo.
* Si no existe varianza en el data set, la técnica de Baggin no mejora significativamente el modelo.
* Es recomendable en modelos de alta inestabilidad (data set con mucha varianza). Ejemplo de inestabilidad: el % de error de la predicción de fraudes de enero, es muy diferente al de febrero.
* Mientras más inestable es un modelo, mejor será la predicción al usar Bagging.
* Se reduce el overfetting o sobre entrenamiento de modelos. Esto porque los modelos no pueden sobre aprender o memorizar ya que ninguno tiene todos los datos de entrenamiento.
* Mejora la predicción, ya que lo que no detecta un modelo lo detectan los otros.
* Reduce el ruido de los outliers, ya que los outliers no pueden estar presenten en todos los modelos.
* No mejora significativamente las funciones lineales, ya que el ensamble de una función lineal da como resultado otra función linear.

## Desarrollo

Se trabajara con los datos de clasificación de vinos de la clase de AED, para predecir la clase de vino según las demás variables

1. Importar las librerías necesarias

*# importamos la libreria pandas y matplotlib, numpy*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*import numpy as np*

*from itertools import cycle*

*# preprocessing module. This contains utilities for scaling, transforming, and wrangling data*

*from sklearn import preprocessing*

*#this module contains many utilities that will help us choose between models.*

*from sklearn.model\_selection import train\_test\_split*

*# importamos libreria scikit para el clasificador del random forest*

*from sklearn.ensemble import RandomForestClassifier*

*# Import evaluation metrics Python*

*from sklearn.metrics import confusion\_matrix*

*from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score*

*from sklearn.metrics import roc\_curve, auc, roc\_auc\_score, classification\_report*

*#import a way to persist our model for future use*

*from sklearn.externals import joblib*

*from scipy import interp*

*from collections import OrderedDict*

*# seteamos el objeto random a 0*

*np.random.seed(0)*

1. Importar el conjunto de datos

*data = pd.read\_csv("Clasificacion.txt", sep="\t")*

*data.head(3)*

1. Separar las variables X y Y

*y = data["Clase Vino"]*

*x = data.drop(["Clase Vino","ID Caso"], axis=1)*

1. Generar un dataset para entrenamiento y otra para test

*# Funcion de Scikit-Learn's para dividir el dataset en conjunto de entrenamiento y prueba*

*x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,*

*test\_size=0.2,*

*random\_state=123)*

1. Crear el objeto que lleva los parámetros para aplicar el modelo de RandomForest

*# Define los parametros para el clasificador de random forest*

*clf = RandomForestClassifier(n\_jobs=2, random\_state=0, max\_depth=5)[[2]](#footnote-2)*

1. Ajuste el modelo con los datos de entrenamiento

*# entrenamos el clasificador para que tome las variables (características) de entrenamiento y aprenda a relacionarse a las especies (y)*

*clf.fit(x\_train, y\_train)*

1. Aplicar el modelo a los datos de prueba y crear un vector con los resultados predichos y el vector de probabilidades

*y\_pred = (clf.predict(x\_test))*

*y\_score = clf.fit(x\_train, y\_train).predict\_proba(x\_test)*

1. Revisar precisión de clasificación con matrices de confusión

*# creamos una matriz de confusion*

*pd.crosstab(y\_test, y\_predict, rownames=['Actual'], colnames=['Predicted'])*

También se puede hacer con: *print(confusion\_matrix(y\_test, y\_predict))*

1. Revisar medidas de error del modelo

*# Imprime medidas de error del modelo*

*print("R^2: " + str(round(r2\_score(y\_test, y\_predict),3)))*

*print("MSE: " + str(round(mean\_squared\_error(y\_test, y\_predict),3)))*

*print("Accuracy: " + str(round(accuracy\_score(y\_test, y\_predict),3)))*

1. Verificar la importancia de las variables en el modelo

*# variables más importantes en el modelo*

*feature\_importances = pd.Series(clf.feature\_importances\_, index=x.columns)*

*feature\_importances.sort\_values(axis=0, inplace= True, ascending=False)*

*print(feature\_importances)*

1. Graficar en barras las variables importantes

*feature\_importances.plot(kind="barh")*

1. Generar la curva ROC del modelo

*y\_roc = np.array(y\_test==y\_predict)*

*fpr, tpr, \_ = roc\_curve(y\_roc, y\_score)*

*auc = round(roc\_auc\_score(y\_roc, y\_score),3)*

*plt.figure()*

*lw = 2*

*plt.plot(fpr,tpr,color='darkorange',*

*lw=lw, label="ROC curve (area = " + str(auc))*

*plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')*

*plt.xlim([0.0, 1.0])*

*plt.ylim([0.0, 1.05])*

*plt.xlabel('False Positive Rate')*

*plt.ylabel('True Positive Rate')*

*plt.title('Receiver operating characteristic example')*

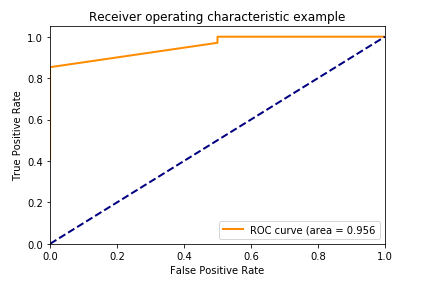
*plt.legend(loc="lower right")*

*plt.show()*

*fpr, tpr, threshs = confusion\_matrix(y\_test, y\_pred)*

*print(fpr, tpr, threshs)*

Debe obtener un resultado como este



Actividad: los métodos Arbol de decisión y random forest, indique sus observaciones.(puede incluir tablas comparativas si es el caso)

1. http://apuntes-r.blogspot.com.co/2014/12/bagging-para-mejorar-un-modelo.html [↑](#footnote-ref-1)
2. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier [↑](#footnote-ref-2)