# UNIVERSIDAD SANTO TOMÁS

## PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

Acreditación Institucional Internacional

OTORGADA POR EL IAC CINDA ACUERDO 55 DEL 9 DE MAYO–VIGENCIA 5 AÑOS

ACREDITACIÓN INSTITUCIONAL DE ALTA CALIDAD MULTICAMPUS

Res. MEN No. 08456 del 28 de enero de 2016

Vigencia por seis años

QS STARS RATED FOR EXCELLENCE

ISO 9001 icontec SC4289–I

CERTIFIED IQNet MANAGEMENT SYSTEM

UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

**Faculty:** Systems engineer

**Course:** Deep Learning

**Topic:** **Sklearn – Arboles de decisión**

---

**Professor:** Luis Fernando Castellanos Guarin

**Email:** Luis.castellanosg@usantoto.edu.co

**Phone:** 3214582098

# *Tipos de modelos de I.A Existen*

- ~~*Regresión lineal*~~
- ~~*Regresión logística.*~~
- ***Árboles de decisión (clasificación y regresión).***
- *K-means*
- *Redes bayesianas*
- *Máquinas de vectores soporte (VSM)*
- ***Deep learning***

# CONTENIDO

1. Que son los arboles de decisión.
2. Clasificación vs predicción
3. Dataset de flor de iris
4. Pasos para árbol de clasificación flor de iris.
5. Dataset de titanic
6. Pasos para árbol de clasificación del titanic
7. Arboles de decisión para regresión

¡Siempre hacia lo alto!

# Arboles de clasificación vs Regresión

Árboles CHAID (Chi-square Automatic Interaction Detector).

Es la conclusión de una serie de métodos basados en el detector automático de interacciones (AID) de Morgan y Sonquist. Es un método exploratorio útil para identificar variables importantes y sus interacciones enfocadas a la segmentación y a los análisis descriptivos.

**Regresión**

**Árboles CART** (Classification and Regression Tree)

- Predecir los volúmenes de ventas el próximo año.
- Definir el mejor precio para un nuevo producto que se lanzara al mercado.
- Predecir los valores de los arriendos de inmuebles en una ciudad para el próximo año.

**Clasificación**

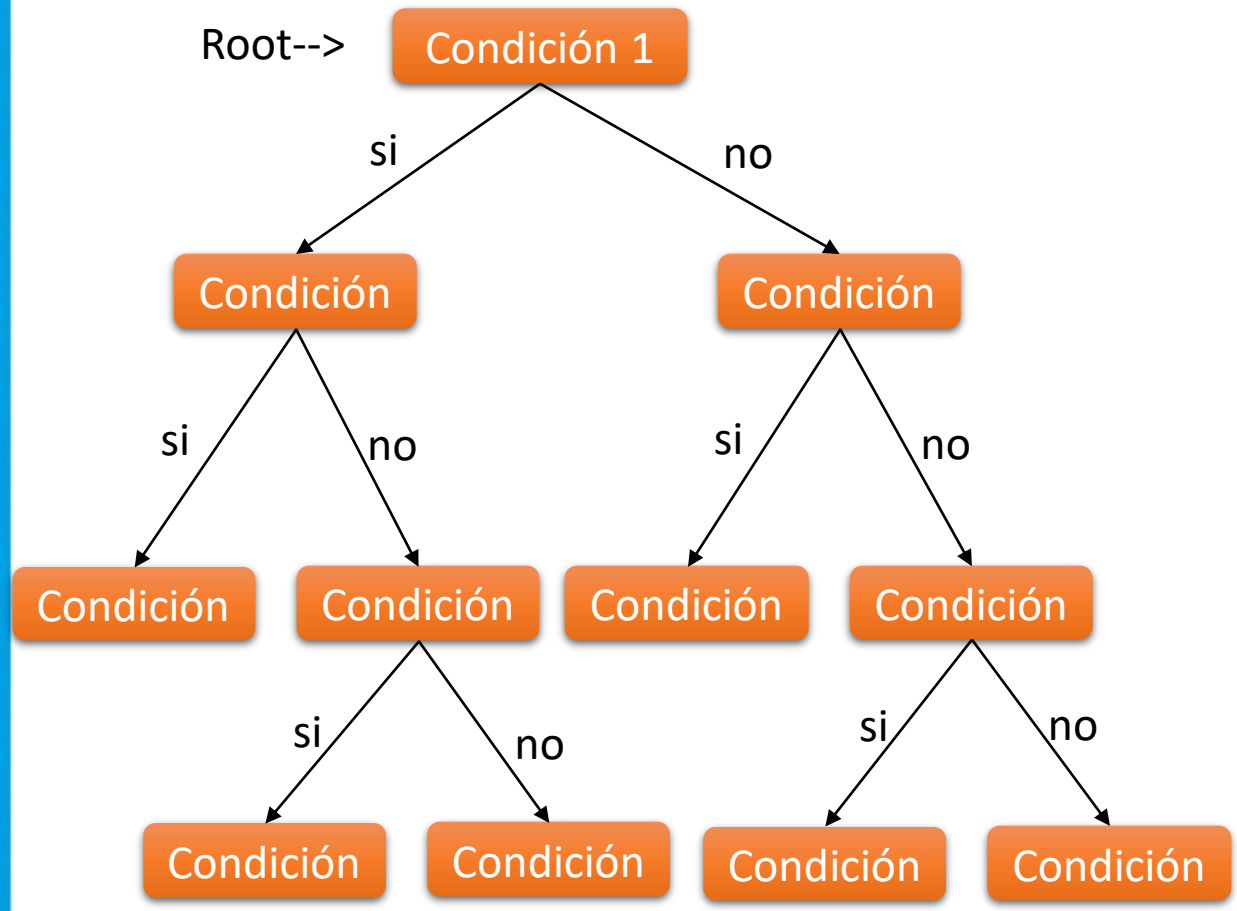
**Árboles QUEST** (Quick, Unbiased, Efficient, Statistica Tree).

- Seleccionar que hacer el fin de semana (ir a cine, restaurante, piscina, plan asado con los amigos)
- Definir si un cliente será “buena paga” o no
- Definir si un proveedor entregara la mercancía a tiempo o no
- Determinar si un paciente con principios de diabetes puede o no agravar su enfermedad.

Los arboles de decisión son algoritmos de aprendizaje supervisado:

**Train vs test**

# Árboles de decisión - clasificación

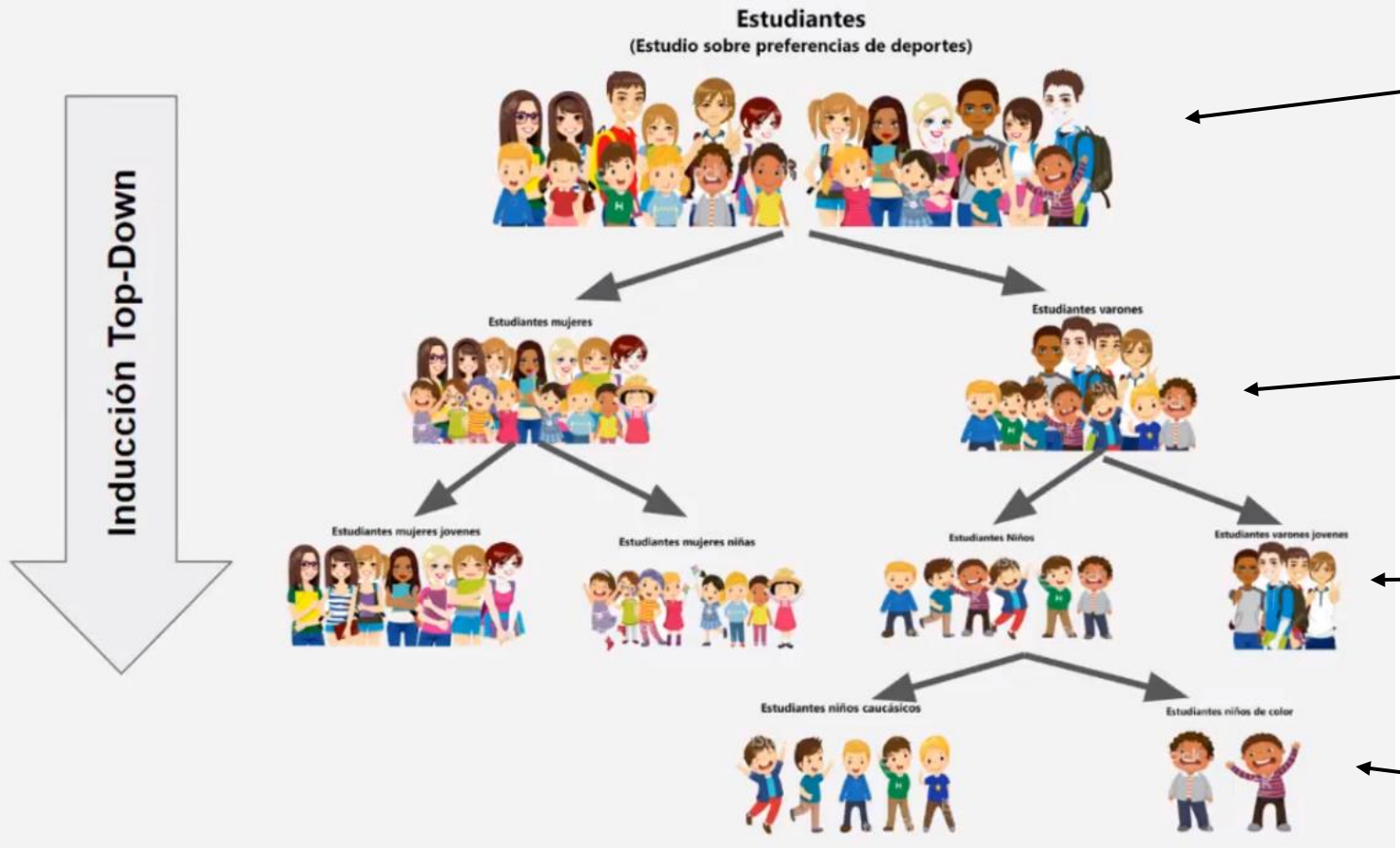



Llueve? => lleva paraguas.
¿Soleado? => lleva gafas de sol.
¿estoy cansado? => toma café.

decisiones del tipo IF(?) THEN ELSE

Los arboles de decisión son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos de aprendizaje supervisado más utilizados en machine learning y pueden realizar tareas de **clasificación o regresión** (acrónimo del inglés **CART**). La comprensión de su funcionamiento suele ser simple y a la vez muy potente.

# Árboles de decisión - clasificación




Grupo de estudiantes (hombres **y** mujeres)

Separamos en dos grupos por separados (hombres **vs** mujeres)

Separamos entre jóvenes vs niños

Separamos entre razas Caucásicos vs de color

¡Siempre hacia lo alto!

# Árboles de decisión - clasificación

Debemos crear un árbol de decisión para clasificar correctamente la variedad de la flor *iris* a partir de 150 muestras donde se tienen la información de:

- Ancho y largo de los pétalos
- sépalos.

Hay tres variedades de flor *iris*:

- 50 iris setosa
- 50 iris versicolor
- 50 iris virginica

Iris setosa

Iris versicolor

Iris virginica

# Árboles de decisión - clasificación

**pasos:**

1. **Cargar librerías**
2. **Cargar dataset**
3. **Explorar datos**
4. **Convertir los datos en un Dataframe** (facilita la visualización)
5. **Separar los datos** (train y test)
6. Crear instancia de algoritmo (árbol de decisión)
7. **Entrenar el algoritmo**
8. **Predecir valores**
9. **Calcular la exactitud del modelo**
10. Graficar el árbol
11. Optimizar el árbol o no?

# Árboles de decisión - clasificación

1. Cargar librerías

```
import numpy as np
import pandas as pd

from sklearn.datasets import load_iris                  #dataset sobre la flor iris (uso libre)
from sklearn.model_selection import train_test_split    #clase para dividir dataset (train y test)
from sklearn.tree import DecisionTreeClassifier         #clase que permite implementar un arbol de desición
import matplotlib.pyplot as plt                         #libreria para generar graficos
```

2. Cargar dataset

```
db_iris = load_iris()
```

3. Explorar datos

```
#imprimiento la descripcion del dataset
print(db_iris.DESCR)
#imprimiento datos del target
print(db_iris.target)
```

4. Convertir los datos en un Dataframe

```
df_iris = pd.DataFrame(db_iris.data, columns=db_iris.feature_names)
df_iris['target'] = db_iris.target
df_iris.head()
```

# Árboles de decisión - clasificación

5. Separar los datos (train y test)

```
#dividiremos el dataset en  75% (train) y 25% (test), estos valores estan por defecto
X_train, X_test, Y_train, Y_test = train_test_split(df_iris[db_iris.feature_names], df_iris['target'], random_state=0)
```

6. Crear instancia de algoritmo (árbol de decisión)

```
arbol_clasi = DecisionTreeClassifier(max_depth = 2, random_state = 0)
```

7. Entrenar el algoritmo

```
arbol_clasi.fit(X_train, Y_train)
```

8. Predecir valores

```
arbol_clasi.predict(X_test[0:10])
```

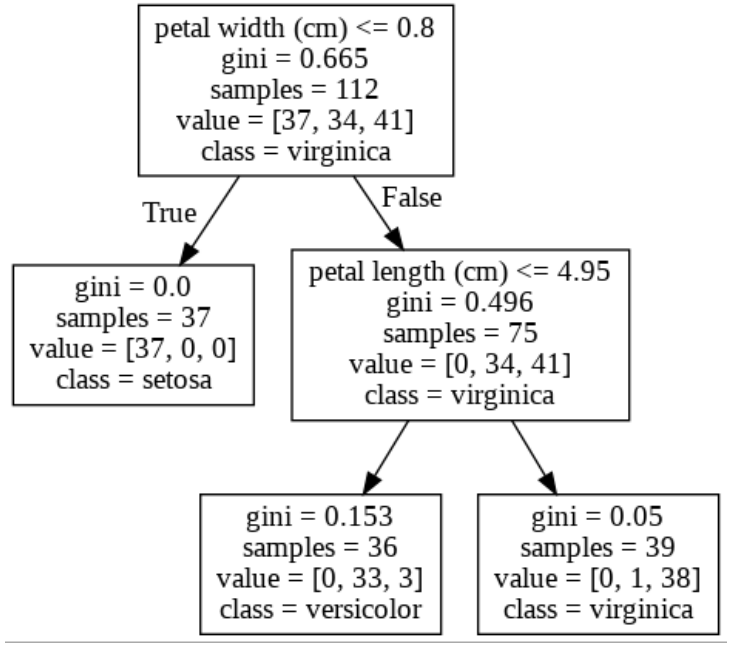
9. Calcular la exactitud del modelo

```
accuary = arbol_clasi.score(X_test, Y_test)
print(accuary)
```

# Árboles de decisión - clasificación

9. Graficar el árbol

```
#para graficar  Iris-Setosa (0), - Iris-Versicolour (1), - Iris-Virginica (2)
class_names_list=list(['setosa','versicolor','virginica'])
from sklearn.tree import export_graphviz
from pydotplus import graph_from_dot_data
dot_data = export_graphviz(arbol_clasi,feature_names=db_iris.feature_names, class_names=class_names_list)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')
```




condición: si es un nodo donde se toma alguna decisión

- gini: es una medida de impureza (entropía ).
- samples: número de muestras que satisfacen las condiciones necesarias para llegar a este nodo
- value: cuántas muestras de cada clase llegan a este nodo
- class: qué clase se le asigna a las muestras que llegan a este nodo

# Árboles de decisión - clasificación

10. Optimizar el árbol o no?

La profundidad del árbol que definimos fue de 2 y que pasa si aumentamos la profundidad (mejorar o empeorara)

Primero, para crear el árbol el algoritmo encuentra que unas características son más importantes que otras...cuales son?

```
#creamos un dataframe de panda (mejora la visualización/administración de los datos)
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(arbol_clasi.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False)
importances.head(10)
```

# Árboles de decisión - clasificación

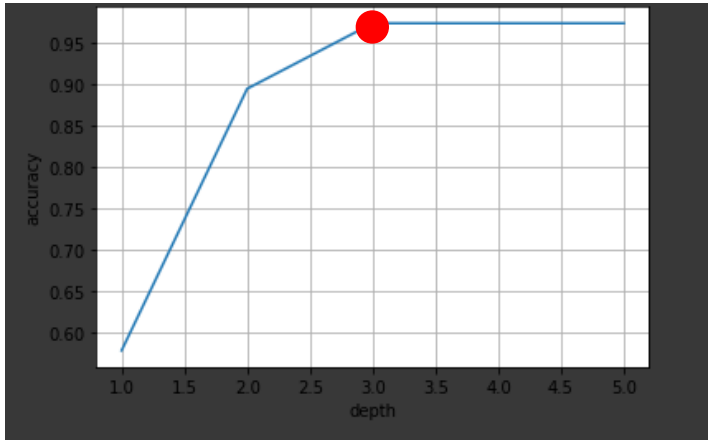
Segundo: probemos con diferentes niveles de profundidad y miramos que tan bueno es el modelo resultante de aplicar el algoritmo

```
#la profundad de un árbol la medimos con max_depth
# creamos una lista con profundidades de 1 a 6
max_depth_lista = list(range(1, 6))
# creamos un listado de resultados de exactitud
accuracy = []
for depth in max_depth_lista:

    arbol_clasi = DecisionTreeClassifier(max_depth = depth, random_state = 0)
    arbol_clasi.fit(X_train, Y_train)
    score = arbol_clasi.score(X_test, Y_test)
    accuracy.append(score)
```

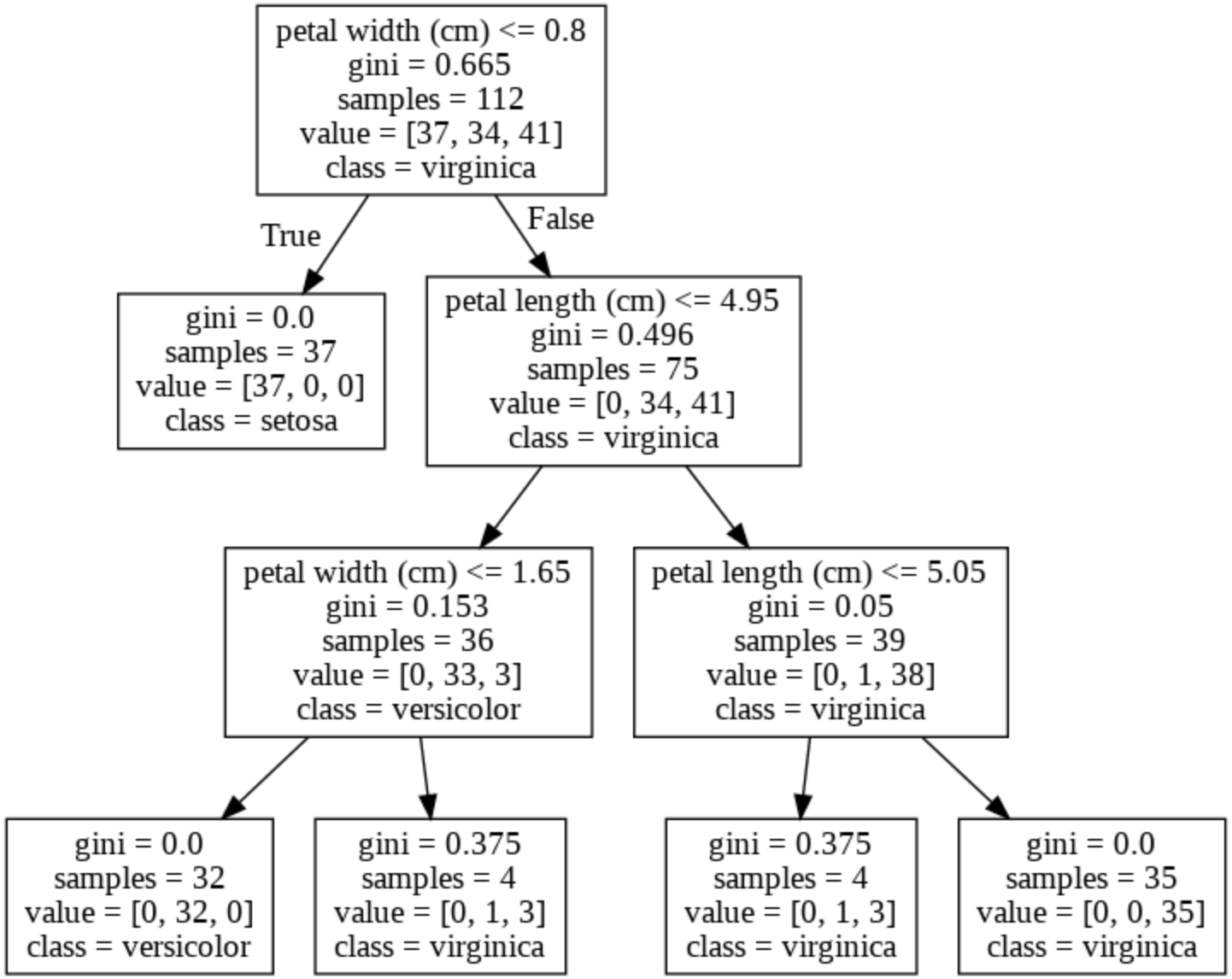
creamos un grafico para ver los resultados

```
plt.plot(max_depth_lista,accuracy)
plt.ylabel('accuracy')
plt.xlabel('depth')
plt.grid(True)
plt.show()
```



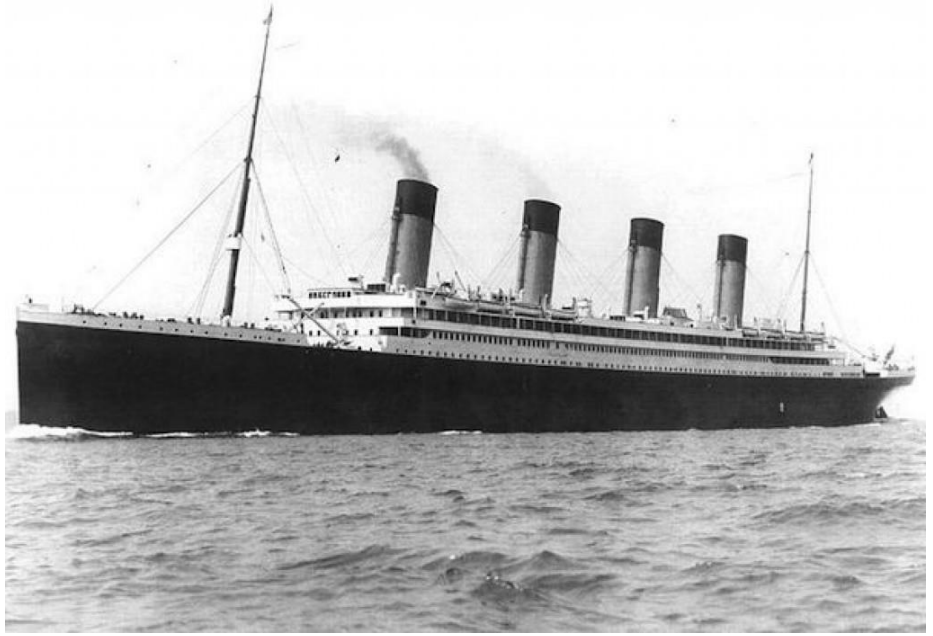
# Árboles de decisión - clasificación

Árbol mejorado:



# Árboles de decisión – titanic (quien sobrevive)

Crear un árbol de decisión que permita conocer quien sobrevivió al naufragio del Titanic.



En el Github:
luisFernandoCastellanosG/Machine_learning/Databaset_para_trabajar_sklearn

Esta disponible la base de datos:
dataset_titanic.csv

El cvs fue tomado de:
https://www.kaggle.com/c/titanic/data

# Árboles de decisión – titanic (quien sobrevive)

**pasos:**

1. Cargar librerías
2. Cargar dataset (csv) en un data frame
3. **Explorar datos**
4. Normalizamos el dataset (Eliminando datos que no son útiles, texto a números, nulos a ceros)
5. Separar los datos (train y test)
6. Crear instancia de algoritmo (árbol de decisión)
7. Entrenar el algoritmo
8. Predecir valores
9. Calcular la exactitud del modelo
10. **Graficar el árbol**
11. Optimizar el árbol o no?

# Árboles de decisión – titanic (quien sobrevive)

**pasos:**

1. Cargar librerías

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split    #clase para dividir dataset (train y test)
from sklearn.tree import DecisionTreeClassifier         #clase que permite implementar un arbol de desición
from sklearn.metrics import accuracy_score              #clase para generar la exactitud del modelo (accuracy)
import matplotlib.pyplot as plt                         #libreria para generar graficos
```

2. Cargar dataset (csv)

```
#podemos cargar el CSV directamente desde GITHUB con el raw
df = pd.read_csv("https://raw.githubusercontent.com/luisFernandoCastellanosG/Machine_learning/master/Databaset_para_trabajar_sklearn/dataset_titanic.csv")
df.head()
```

3. **Explorar datos**

```
#nombres de las columnas
df.columns
#descripción de información del df
df.describe()
# información del tipo de datos que tiene el df
df.info()
#valores nulos
df.isnull().sum()
```

# Árboles de decisión – titanic (quien sobrevive)

## 3. Explorar datos

```
#mostrando datos, con porcentajes
datos=df.Survived.value_counts(normalize = True)
print(datos)
plt.pie(datos, labels=["No","Si"], autopct="%0.1f %%")
plt.title("sobrevivieron - cuenta total")
plt.show()
```

```
plt.title("sobrevivientes - Male VS female")
datos=df.Sex[df.Survived == 1].value_counts(normalize = True)
print(datos)
plt.pie(datos, labels=["female","men"], autopct="%0.1f %%")
plt.show()
```

```
#sobrevivientes por clase o ticket (barras)
fig = plt.figure(figsize=(10,5))
#colors bgrcmykw
df.Pclass[df.Survived == 1].value_counts(normalize = True).plot(kind="barh", alpha=0.5)
plt.title("Sobrevivientes por clase de ticket")
plt.show()
```

```
#clases vs edad
fig = plt.figure(figsize=(20,10))
plt.title("Sobrevivientes por clase y edad")
for t_class in [1,2,3]:
    df.Age[df.Pclass == t_class].plot(kind="kde")

plt.legend({"primera clase", "segunda clase", "tercera clase"})
plt.show()
```

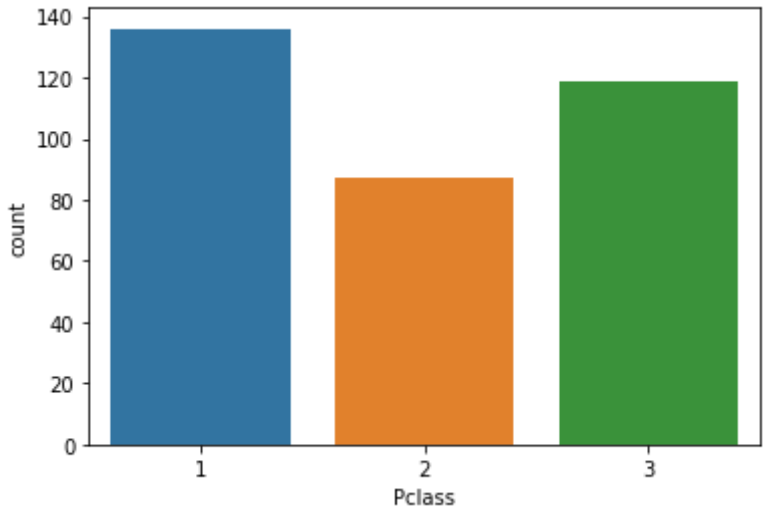
# Árboles de decisión – titanic (quien sobrevive)

## 3. Explorar datos

```
import seaborn as sns
sns.pairplot(df)
```

```
#crear un histograma de 0 a 80 de 5 en 5
sns.distplot(df.Age,bins = np.arange(0,80,5), kde = False)
plt.show()
```

```
#distribución por clases
sns.countplot(x="Pclass", data=df[df["Survived"] == 1])
plt.show()
```



# Árboles de decisión – titanic (quien sobrevive)

4. Normalizamos el dataset (Eliminando datos que no son útiles, texto a números, nulos a ceros)

   Las siguientes variables del dataset no aportan valor alguno al estudio

   - Cabin
   - PassengerId
   - Name
   - Ticket
   - Embarked (puerto de embarque)

```
df.drop(['Cabin','PassengerId','Name','Ticket','Embarked'], axis=1,inplace=True)
df.dropna(inplace=True)
df.head()
```

reemplazando palabras por letras

```
#creamos un diccionario con los valores originales y los valores de reemplazo
a = {"male" : "M", "female" : "F"}
df["Sex"] = df["Sex"].apply(lambda x:a[x])
df["Sex"].head()
```

Convertimos datos de texto a numéricos

```
from sklearn import preprocessing
#cambiamos las palabras (male, female) por (1, 0)
le = preprocessing.LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex'])
```

reemplazando nulos con ceros

```
df["Age"] = df["Age"].fillna(0)
```

# Árboles de decisión – titanic (quien sobrevive)

5. Separamos los datos train – test

*X= (sexo + clase) Y= sobrevivio*

```
X = df[['Pclass','Sex']]
y = df['Survived']
#cambiamos la proporción de 70%-train y 30%-test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

6. Instanciar un árbol de decisión

```
arbol = DecisionTreeClassifier()
```

7. Entrenamos el modelo

```
arbol.fit(X_train, y_train)
```

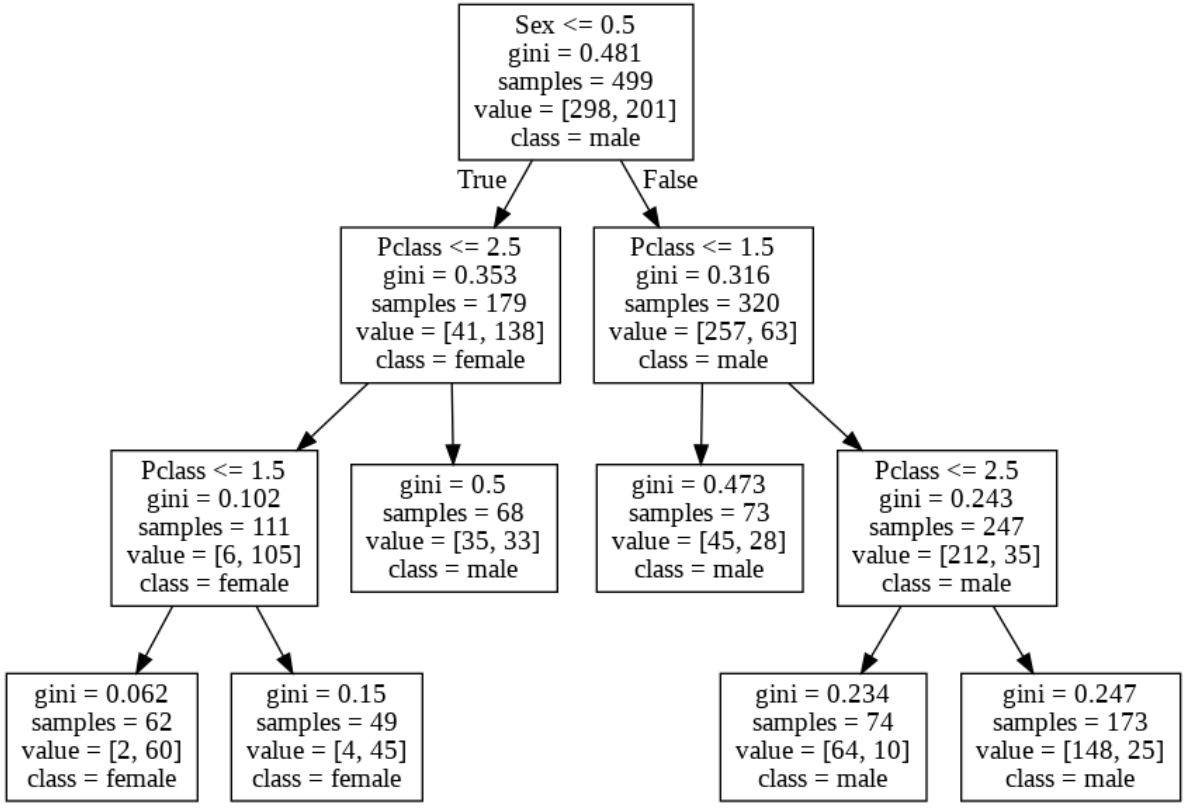
8. Realizamos predicciones

```
# Predecimos sobre nuestro set de entrenamieto
y_train_pred = arbol.predict(X_train)
# Predecimos sobre nuestro set de test
y_test_pred = arbol.predict(X_test)
# Comaparamos con las etiquetas reales
print('Accuracy sobre conjunto de Train:', accuracy_score(y_train_pred,y_train))
print('Accuracy sobre conjunto de Test:', accuracy_score(y_test_pred,y_test))
#Cuando el error de entrenamiento es mayor al del test es porque hay overfitting
```

# Árboles de decisión – titanic (quien sobrevive)

9. Graficamos el árbol

```
#para graficar  Iris-Setosa (0), - Iris-Versicolour (1), - Iris-Virginica (2)
class_names_list=list(['male', 'female'])
from sklearn.tree import export_graphviz
from pydotplus import graph_from_dot_data
dot_data = export_graphviz(arbol,feature_names=['Pclass','Sex'], class_names=class_names_list)
graph = graph_from_dot_data(dot_data)
graph.write_png('arbol_titanic.png')
```



# Árboles de decisión - Regresión

El proceso de resolución de problemas de regresión con árbol de decisión utilizando **Scikit Learn** es muy similar al de clasificación.

Sin embargo, para la regresión usamos la clase **DecisionTreeRegressor** de la biblioteca de árboles.

Además, las matrices de evaluación para la regresión difieren de las de clasificación.

**El resto del proceso es casi el mismo.**

Para la explicación usaremos un conjunto de datos para intentar predecir el consumo de gasolina (en millones de galones) en 48 estados de EE. UU. En función del impuesto a la gasolina (en centavos), el ingreso per cápita (dólares), las carreteras pavimentadas (en millas) y la proporción de la población con un Licencia de conducir.

En el Github:
luisFernandoCastellanosG/Machine_learning/Databaset_para_trabajar_sklearn

Esta disponible la base de datos:
petrol_consumption.csv

Para conocer el detalle de los datos: https://people.sc.fsu.edu/~jburkardt/datasets/regression/x16.txt

# Árboles de decisión - Regresión

**pasos:**

1. Cargar librerías
2. Cargar dataset (csv) en un data frame
3. **Explorar datos**
4. Normalizamos el dataset (Eliminando datos que no son útiles, texto a números, nulos a ceros)
5. Separar los datos (train y test)
6. Crear instancia de algoritmo (árbol de decisión)
7. Entrenar el algoritmo
8. Predecir valores
9. Calcular la exactitud del modelo
10. **Graficar el árbol**
11. Optimizar el árbol o no?

# Underftting y OverFitting

## Underfitting

Entreno al modelo con
1 sóla raza de perro

Muestra nueva:
¿Es perro?

NO

FALLO

La máquina fallará en reconocer al perro por falta de suficientes muestras. No puede generalizar el conocimiento.

El modelo se ajustara para aprender **los casos particulares que le enseñamos** y será incapaz de reconocer nuevos datos de entrada

Cuando sobre-entrenamos nuestro modelo y caemos en el overfitting, nuestro algoritmo estará considerando como válidos sólo los datos idénticos a los de nuestro conjunto de entrenamiento y siendo incapaz de distinguir entradas buenas como fiables si se salen un poco de los rangos ya preestablecidos

## Overfitting

Entreno al modelo con
10 razas de perro color marrón

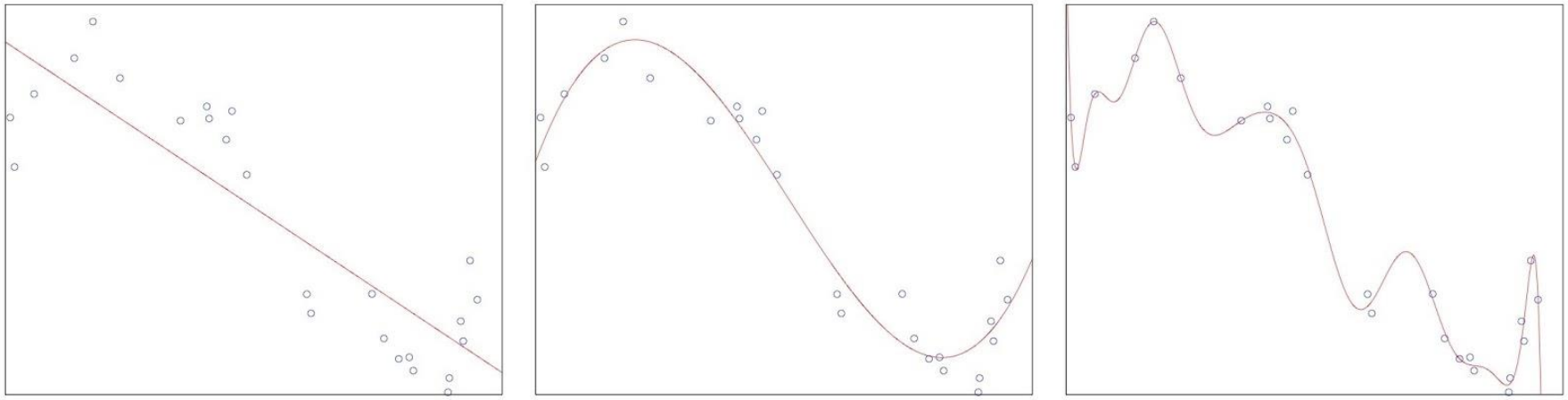
Muestra nueva:
¿Es perro?

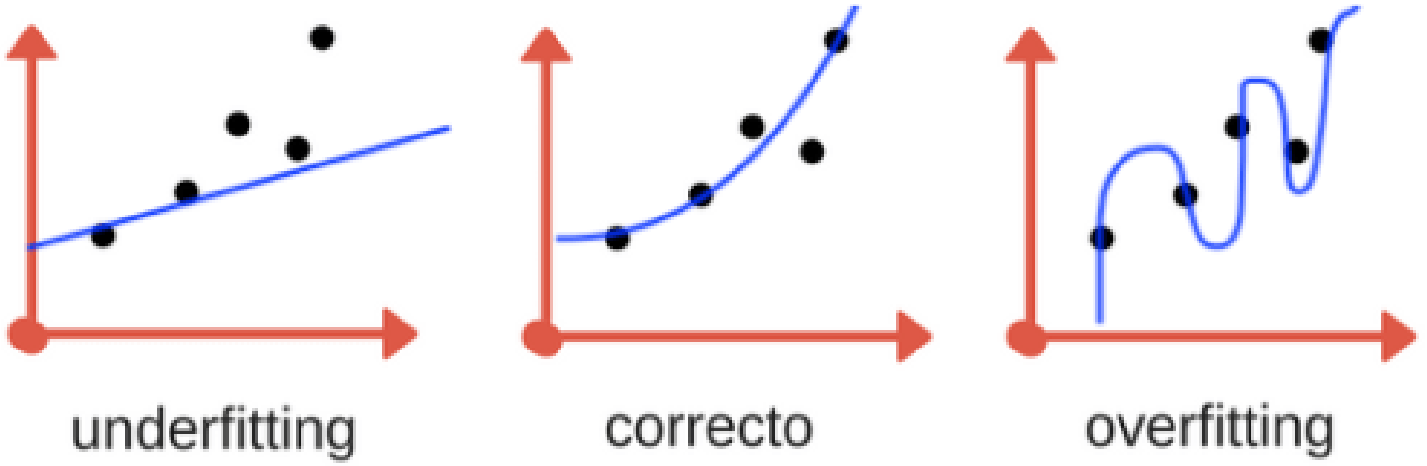
NO

FALLO

La máquina fallará en reconocer un perro nuevo porque no tiene estrictamente los mismos valores de las muestras de entrenamiento.

# Underftting y OverFitting



# Underftting y OverFitting

Estos errores (Underfitting y OverFitting) se pueden prevenir o corregir:

- Aumentando el número de muestras de entrenamiento. Esta "***técnica***" podría parecer el curso de acción más lógico, sin embargo, adquirir mayor cantidad de muestras de entrenamiento no siempre es posible o en algunos casos demanda un capital financiero elevado.
- Mejorar la selección de características con alta correlación
- Bajar la dimensionalidad del descriptor y en consecuencia del modelo, se puede lograr usando algoritmos como:
  - Análisis de componentes principales (PCA): seleccionar características con mayor relevancia.
  - ADABOOST: combina los datos clasificados de forma errónea para darle y se vuelve a entrenar para aumentar los pesos.
- Selección de otros modelos, hasta encontrar el que mejor exactitud logre usando "cross-validation".

¡Siempre
hacia lo alto!

# UNIVERSIDAD SANTO TOMÁS

PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA

SECCIONAL TUNJA

VIGILADA MINEDUCACIÓN - SNIES 1732

## ¡Siempre hacia lo alto!

USTATUNJA.EDU.CO

@santotomastunja