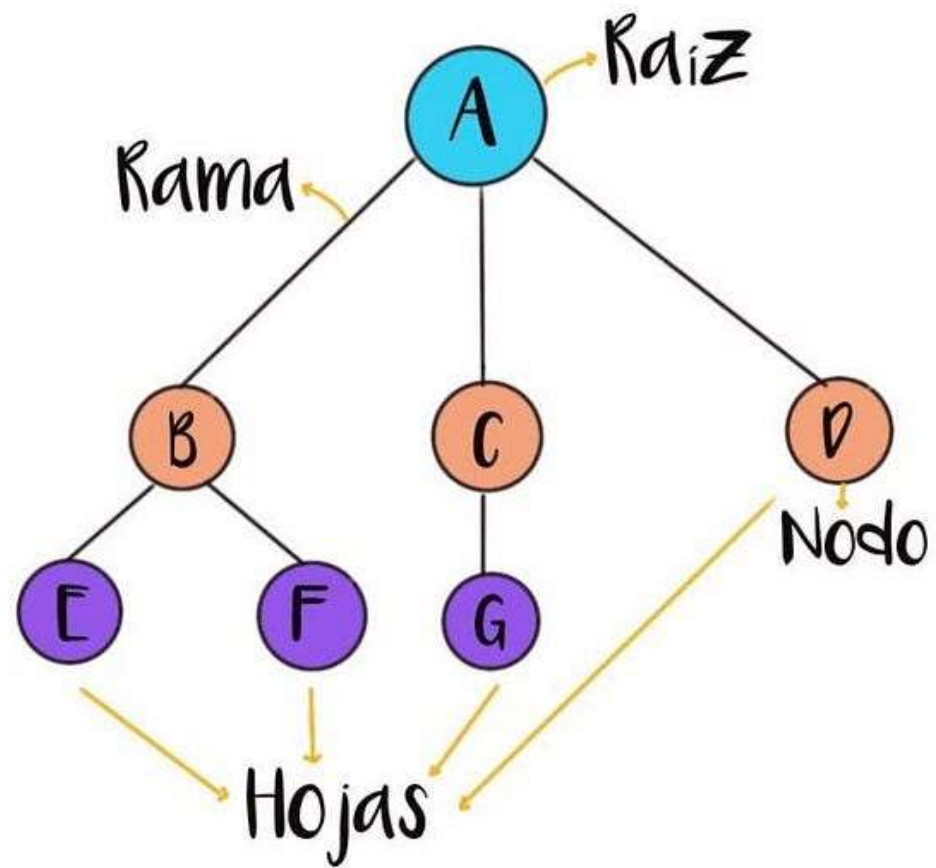


Estructura de un árbol de decisión



Principales Conceptos de la Teoría de la Información

1. Entropía:

- **Definición:** La entropía es una medida de la incertidumbre asociada a una fuente de información. En términos simples, cuantifica cuánta información (o sorpresa) hay en un mensaje. La entropía $H(X)$ para una variable aleatoria X con posibles valores x_1, x_2, \dots, x_n y probabilidades p_1, p_2, \dots, p_n se define como:

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i$$

- **Interpretación:** Si todos los eventos son igualmente probables, la entropía es máxima. Si un evento es seguro, la entropía es cero.



La optimización de un algoritmo tipo árbol se basa en la GANACIA DE INFORMACION



La entropía como una medida de impureza

1. Definiciones:

- p_1 representa la fracción de ejemplos que pertenecen a una clase particular (en este caso, "gatos").
- p_0 se define como $1 - p_1$, que representa la fracción de ejemplos que no pertenecen a esa

La entropía $H(p_1)$ mide la incertidumbre o impureza de la clasificación.

- La entropía $H(p_1)$ se calcula usando la siguiente fórmula:

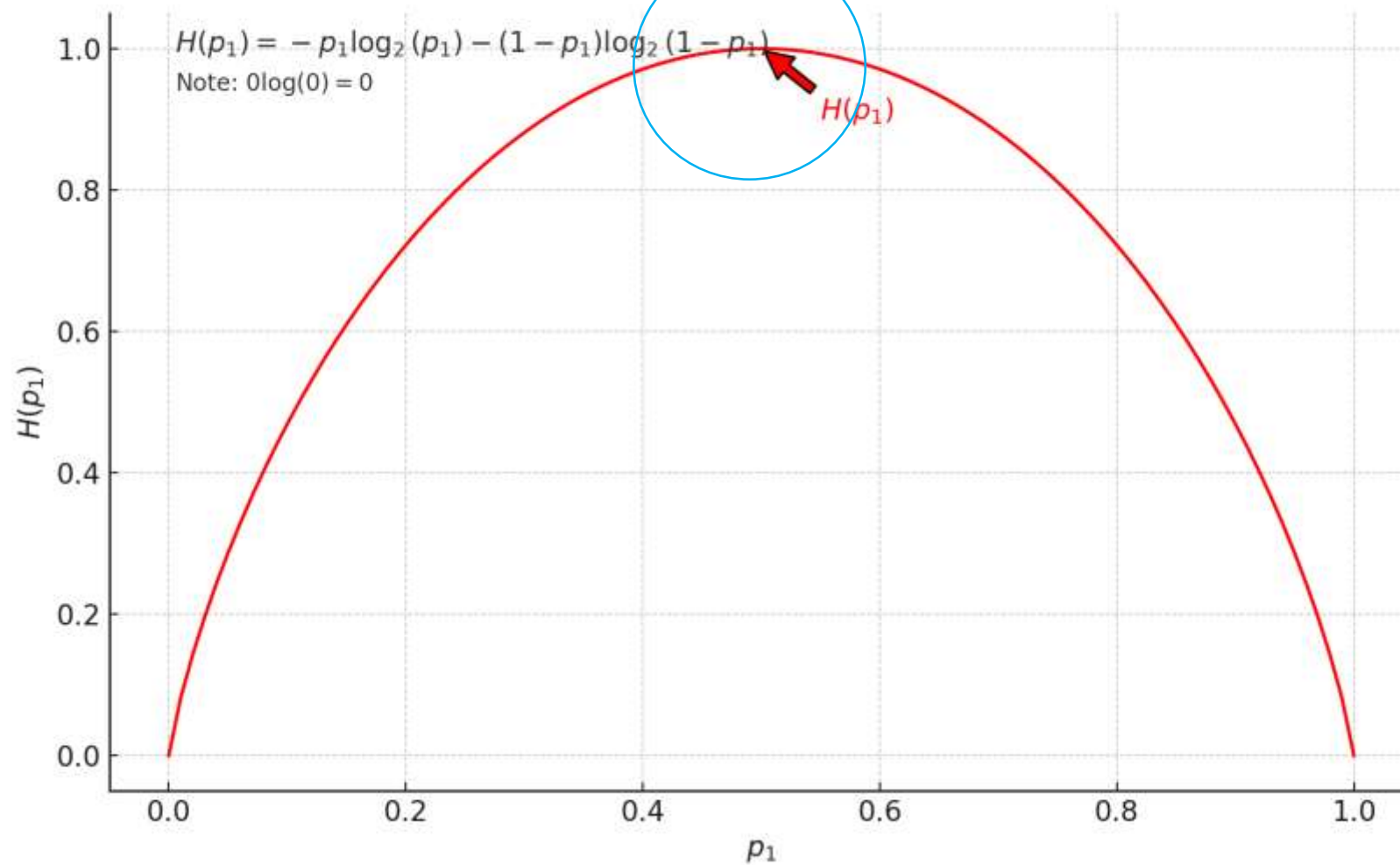
$$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

- Sustituyendo $p_0 = 1 - p_1$ en la fórmula se obtiene:











$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$

"0 log(0) = 0"

Entropy as a measure of impurity

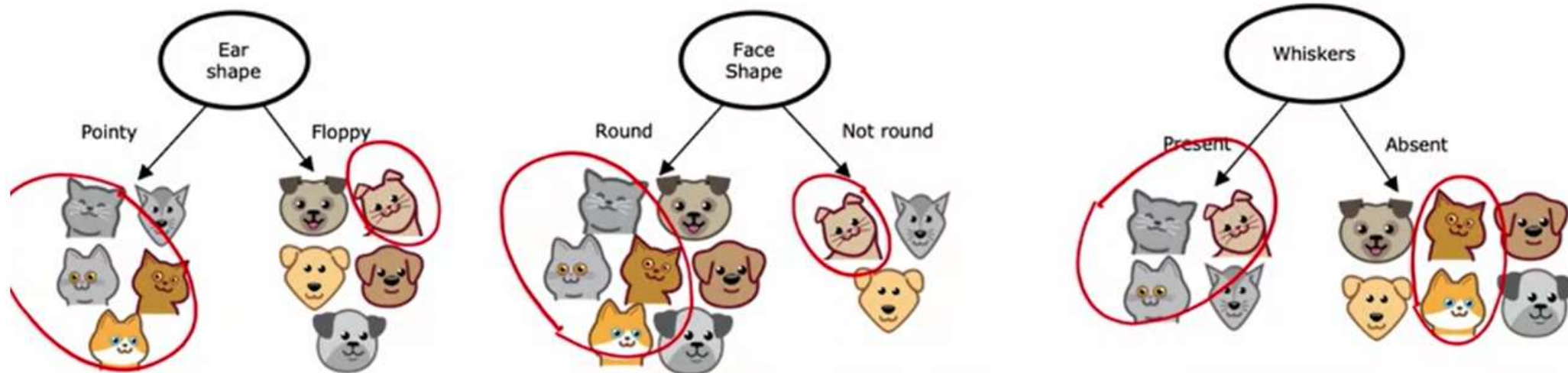


Cat classification example

	Ear shape	Face shape	Whiskers	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Not round	Present	0
	Pointy	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0

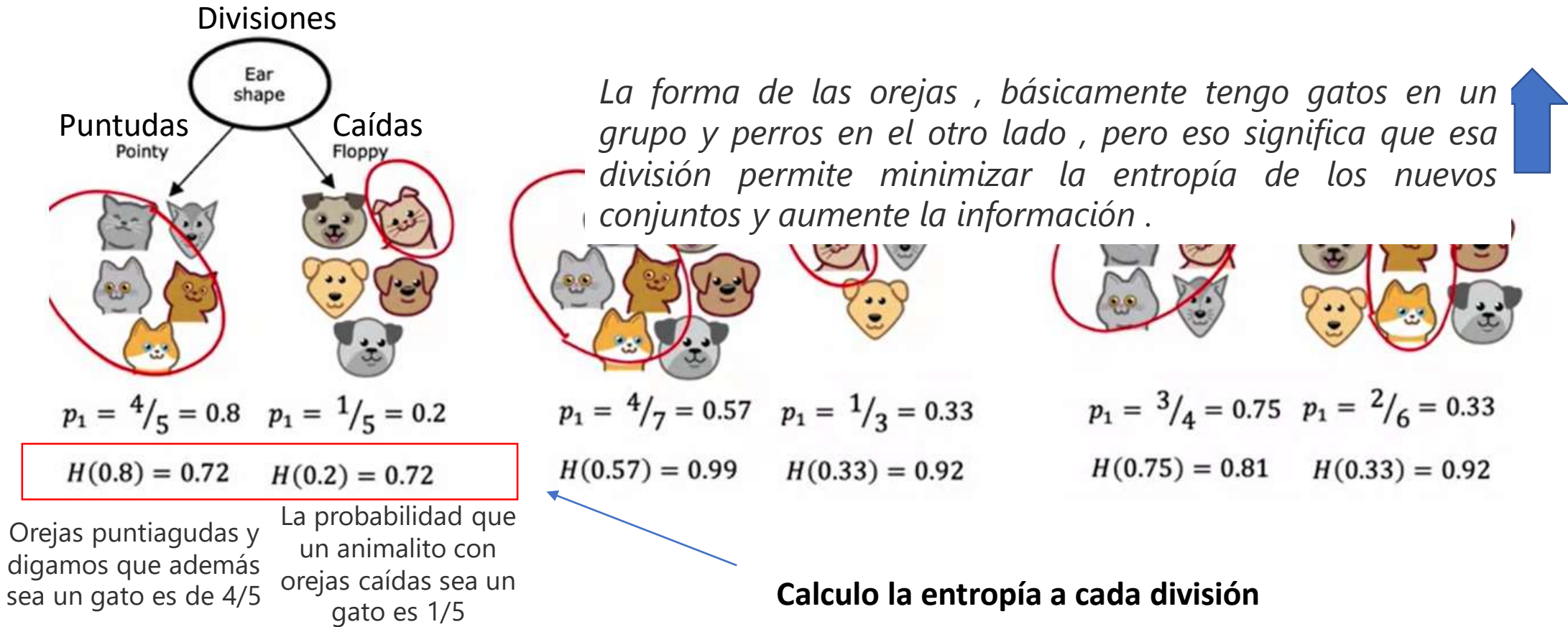
Choosing a split

Diferentes tipos de Split

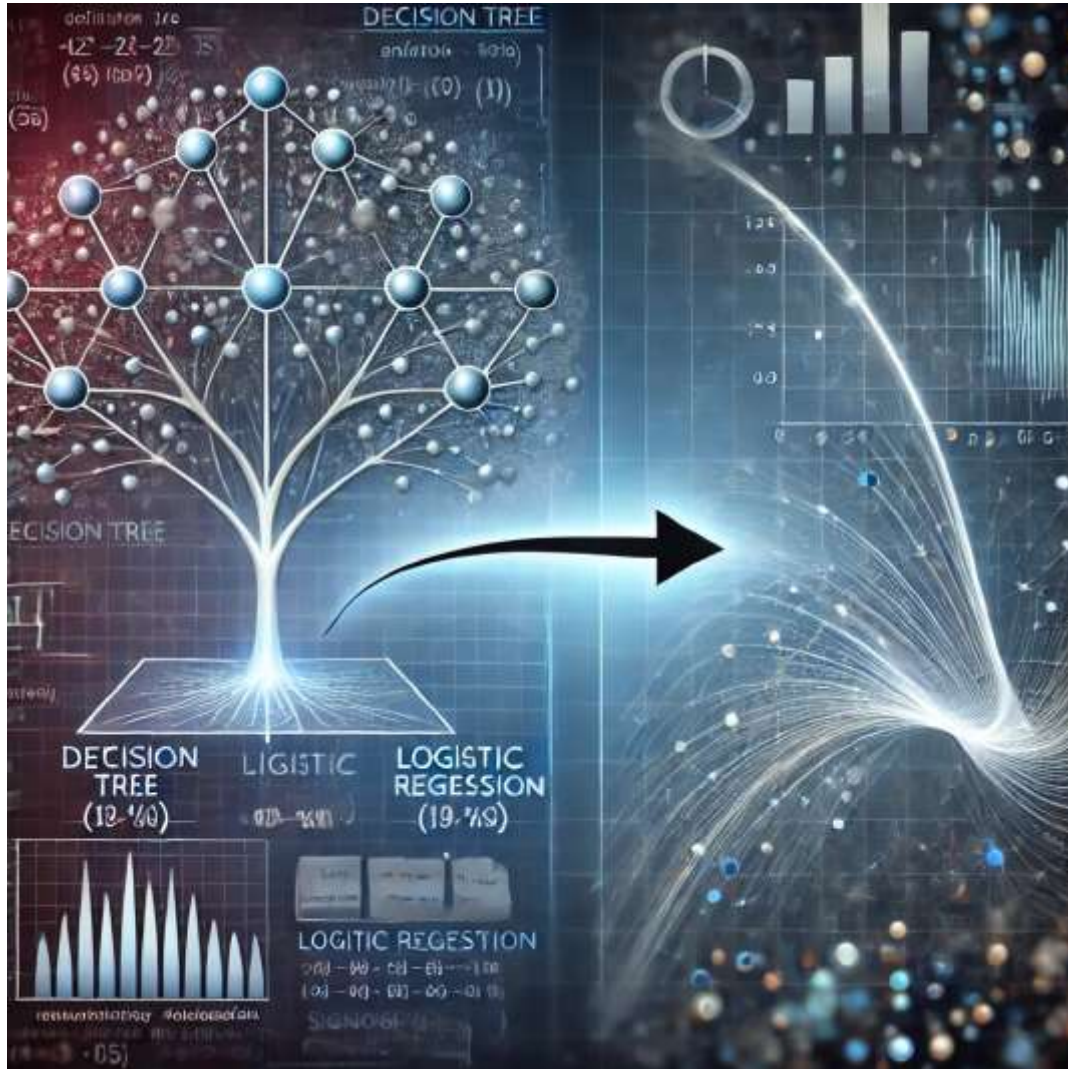


PARTICIONES QUE MINIMICEN LA ENTROPIA DE LOS CONJUNTOS , ES DECIR CADA DIVISION DE CARACTERIZTICAS ME PERMITIRA DISMINUIR LA ENTROPIA

$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$

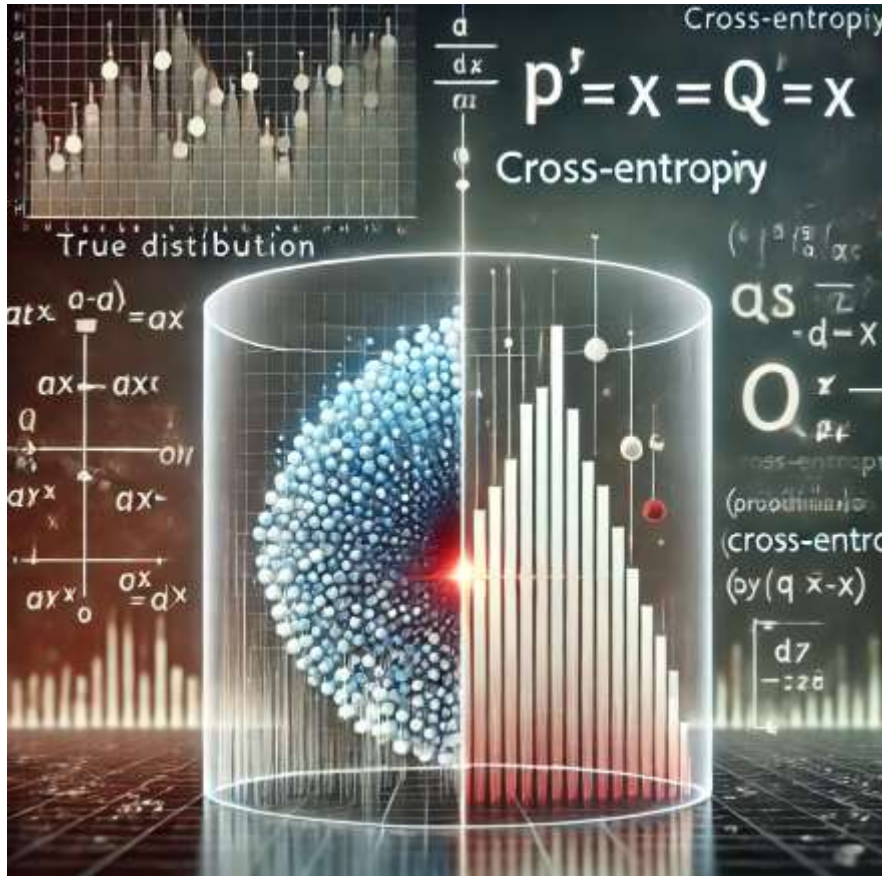


$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$



Ambos enfoques aplican la teoría de la información con el objetivo de reducir la incertidumbre. Esta incertidumbre puede medirse como entropía en un árbol de decisión o como pérdida en una regresión logística. Para mejorar u optimizar un modelo basado en la teoría de la información, ya sea en clasificación mediante regresión logística o en predicción con un árbol de decisión, se busca minimizar la incertidumbre. En ambos casos, el objetivo es reducirla, calculándola ya sea a través de la entropía en un árbol de decisión o de la pérdida en la regresión logística.

La entropía cruzada o ponderada es una métrica utilizada en aprendizaje automático, específicamente en tareas de clasificación, para medir la diferencia entre dos distribuciones de probabilidad: En términos simples, la entropía cruzada cuantifica cuán bien un modelo predice el resultado correcto. Cuanto menor sea el valor de la entropía cruzada, mejor es la predicción del modelo.



Definición Matemática

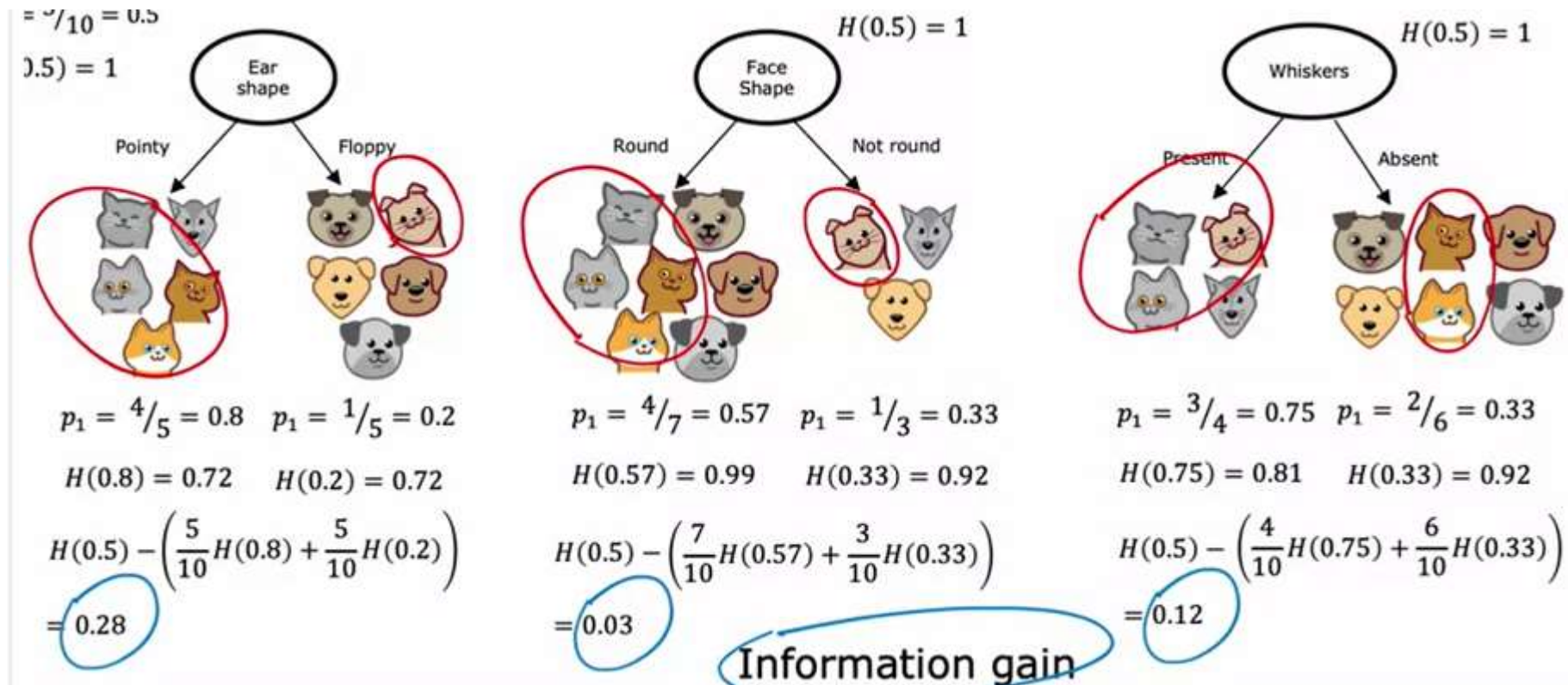
La entropía ponderada se puede calcular como:

$$H_w(p) = - \sum_i w_i \cdot p(x_i) \log(p(x_i))$$

Donde:

- $p(x_i)$ es la probabilidad del evento x_i .
- w_i es el peso asignado al evento x_i , que refleja su importancia relativa.

La ganancia de información se define como la diferencia entre la entropía máxima, obtenida al hacer una división aleatoria, y la reducción de entropía lograda al dividir los ítems según una característica específica. Es decir, mide cuánto se reduce la incertidumbre al hacer una partición basada en una característica en comparación con una división aleatoria.



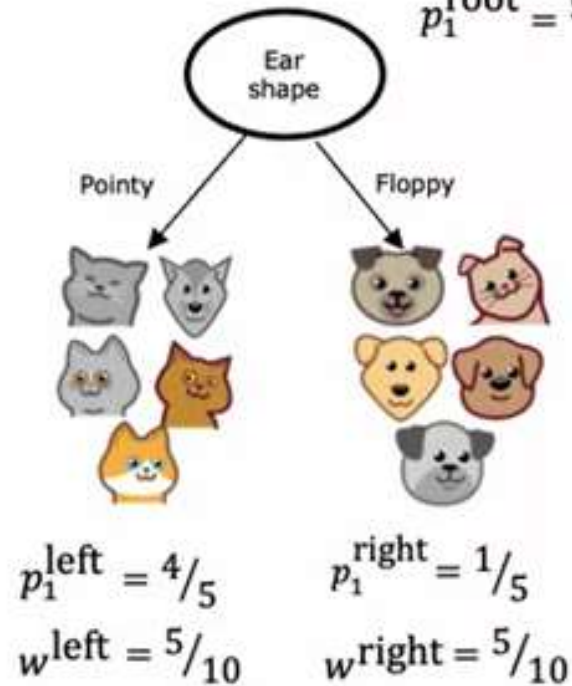
$$H_w(p) = - \sum_i w_i \cdot p(x_i) \log(p(x_i))$$

¿Cuáles van a ser las mejores características para utilizar?

Information Gain



$$p_1^{\text{root}} = 5/10 = 0.5$$



Information gain

$$= H(p_1^{\text{root}}) - \left(w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}) \right)$$



Porque en ese caso dijimos que era 1, porque al principio, antes de hacer las divisiones, se observa que tengo el mismo número de perros y gatos

$$100 = 50 / 50 \quad p_1 = 0.5$$

$$100 = 90 / 10 \quad p_1 = 0.9$$

$$100 = 10 / 100 \quad p_1 = 0.1$$

Information gain

$$= H(p_1^{\text{root}}) - \left(w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}) \right)$$

$$H(p_1) = -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1)$$

```
def information_gain(X, y, left_indices, right_indices):  
    """  
    Here, X has the elements in the node and y is theirs respective classes  
    """  
    p_node = sum(y)/len(y)  
    h_node = entropy(p_node)  
    w_entropy = weighted_entropy(X,y,left_indices,right_indices)  
    return h_node - w_entropy
```

```
information_gain(X_train, y_train, left_indices, right_indices)
```

```
0.2780719051126377
```


Aprendizaje de Árboles de Decisión

- Comienza con todos los ejemplos en el nodo raíz. Donde están todos los items
- Calcula la ganancia de información para todas las características posibles y elige la que tenga la mayor ganancia de información.
$$= H(p_1^{\text{root}}) - (w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}))$$
- Divide el conjunto de datos según la característica seleccionada y crea las ramas izquierda y derecha del árbol. Divido el data SET con base a la característica seleccionada y creo ramas, derecha e izquierda para el árbol (repetir.)
- Sigue repitiendo el proceso de división hasta que se cumplan los criterios de detención:
 - Cuando un nodo es 100% de una clase. 100 datos , llego aun nodo con 30 gatos me identifica completamente q que son gatos- Clasificación con éxito
 - Cuando dividir un nodo resultará en que el árbol exceda una profundidad máxima.
Parámetro limite : Si un árbol se ramifica en exceso el árbol sufre overfitting (grafo : ecuación)
 - Cuando la ganancia de información de divisiones adicionales es menor que un umbral.
 - Cuando el número de ejemplos en un nodo está por debajo de un umbral.

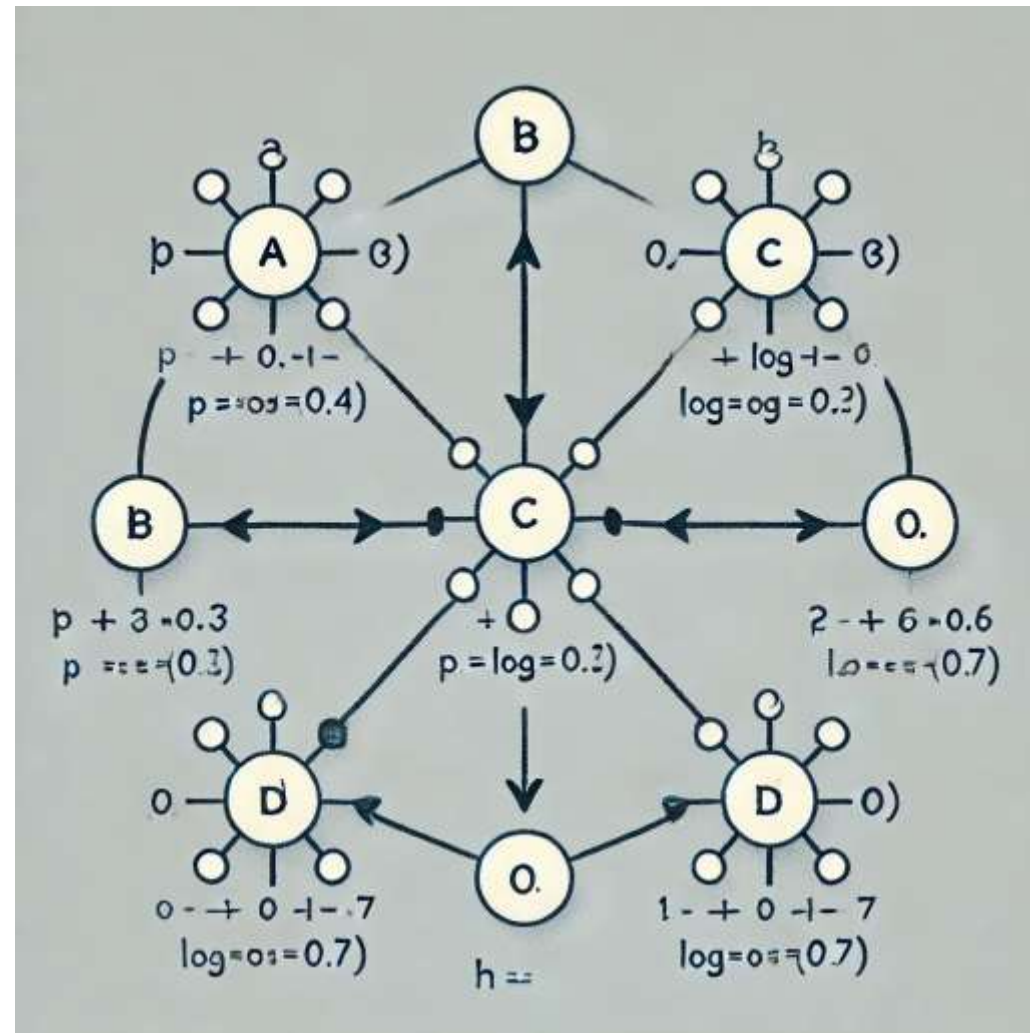



Imagen del grafo dirigido simple con los nodos A, B, C, D, y sus respectivas transiciones etiquetadas con las probabilidades y los pesos. También se incluye la ecuación de la entropía ponderada asociada a este grafo.

¿Por qué es importante el parámetro `max_depth`?

1. **Evitar el sobreajuste (Overfitting):** Un árbol con demasiada profundidad puede ajustarse demasiado a los datos de entrenamiento, capturando tanto las tendencias reales como el ruido aleatorio. Esto puede llevar a un modelo que funciona muy bien en el conjunto de entrenamiento pero que tiene un mal rendimiento en datos nuevos (conjunto de prueba).
2. **Mejorar la generalización:** Al limitar la profundidad del árbol, se puede conseguir un modelo que generalice mejor, es decir, que tenga un buen rendimiento tanto en el conjunto de entrenamiento como en el conjunto de prueba.
3. **Controlar la complejidad:** Árboles más profundos son más complejos y tienden a ser más difíciles de interpretar. Limitar la profundidad puede simplificar el modelo.

python

 Copiar código

```
from sklearn.tree import DecisionTreeClassifier

# Creación del modelo con una profundidad máxima de 5
clf = DecisionTreeClassifier(max_depth=5)

# Entrenamiento del modelo
clf.fit(X_train, y_train)
```

¿Cómo elegir el valor adecuado para `max_depth`?

Conexión entre `log2(n)` y `max_depth`

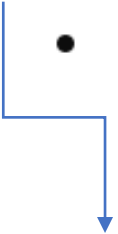
- **Árbol de decisión balanceado:** En un árbol de decisión completamente balanceado, la profundidad del árbol está relacionada con la cantidad de datos a través del logaritmo en base 2. Específicamente, un árbol balanceado con n muestras tendrá una profundidad aproximada de $\log_2(n)$. Este es el mínimo número de divisiones necesarias para separar n muestras en hojas individuales.
 - Ajuste del modelo: `grid_search.fit(X, y)` ajusta el modelo para cada combinación de hiperparámetros en `param_grid`, utilizando validación cruzada para evaluar el rendimiento.
- `max_depth` en valor cercano a $\log_2(n)$ podría hacer que el árbol sea demasiado simple, llevando al subajuste.

Supongamos que tienes 1024 muestras en tu dataset ($n = 1024$):

$$\log_2(1024) = 10$$

```
# Definir el rango de profundidades a probar
max_depths = range(1, 21) # Desde profundidad 1 hasta 20
```

- Sigue repitiendo el proceso de división hasta que se cumplan los criterios de detención:
 - Cuando un nodo es 100% de una clase.
 - Cuando dividir un nodo resultará en que el árbol exceda una profundidad máxima.
 - Cuando la ganancia de información de divisiones adicionales es menor que un umbral.
 - Cuando el número de ejemplos en un nodo está por debajo de un umbral.



La ganancia de información ya no le otorgue más información al modelo, le puedo poner otro parámetro, que es la ganancia mínima de información.

¿Qué es `min_samples_leaf`?

- `min_samples_leaf` es un hiperparámetro que define el número mínimo de muestras que un nodo hoja debe contener después de dividir un nodo. Si una división de un nodo genera nodos hojas con menos muestras que este valor mínimo, esa división no se realiza.

1000 datos ---50 (la ganancia es mínima no itere mas)

Parameter	Sklearn alias	Recommended range
num_boost_round	n_estimators	fixed: high value combine with early stopping
eta	learning_rate	[0.01, 0.3]
max_depth	max_depth	[3, 10]
subsample	subsample	[0.5, 1]
colsample_bytree	colsample_bytree	[0.5, 1]
gamma	gamma	[0, 10]
min_child_weight	min_child_weight	[0, 10]
lambda	reg_lambda	[0, 1]
alpha	reg_alpha	[0, 1]

Subsample= cuantas columnas utiliza en cada división (0.9=90%) Por defecto toma todas características para las Divisiones

Múltiples arboles , cuantas columnas por árbol 0.8 cada árbol con el 80% .Durante el entrenamiento

Tamaño o peso mínimo que toma cada hijo de los nodos
Cada que hace una partición es cuanto es el mínimo que puede ir A cada lado.

¿Qué es `n_estimators`?

- `n_estimators` es un parámetro que especifica el número de árboles (o estimadores) que se construirán en un conjunto de árboles, como en un modelo de Random Forest o Gradient Boosting.

```
# Crear un Random Forest con 100 estimadores
clf = RandomForestClassifier(n_estimators=100, random_state=42)
```


Learning Trees

Decision Tree



Random Forest



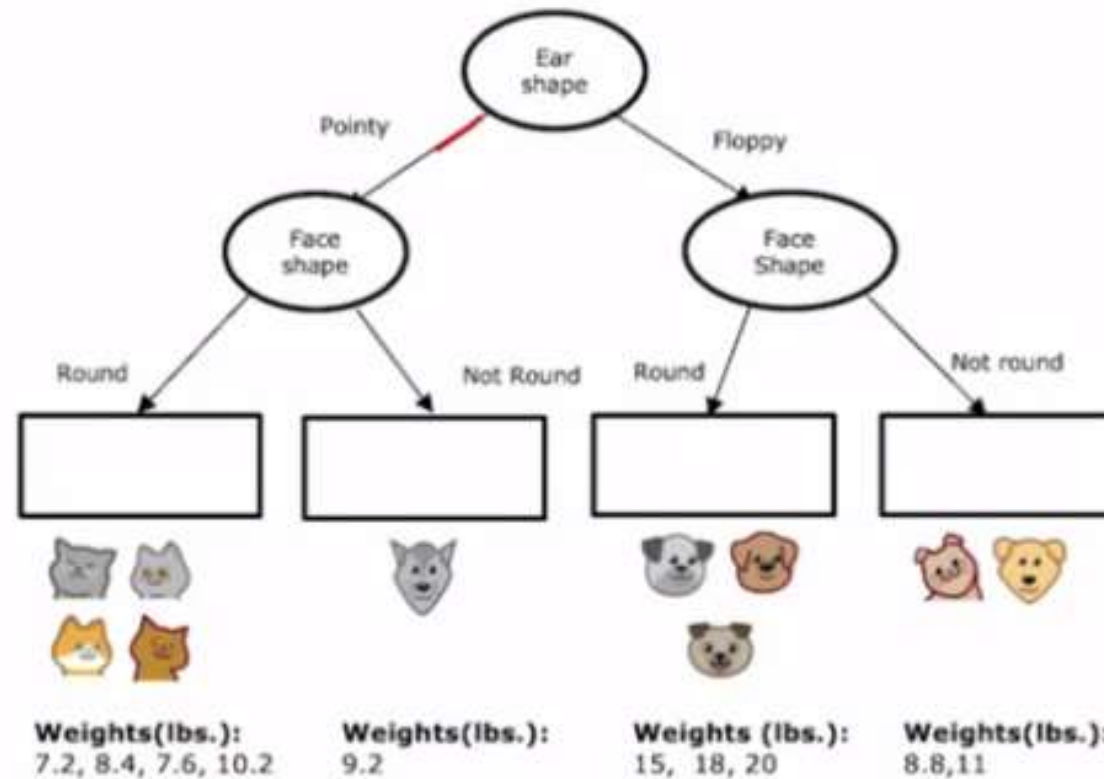
¿Qué es `n_estimators`?

- `n_estimators` es un parámetro que especifica el número de árboles (o estimadores) que se construirán en un conjunto de árboles, como en un modelo de Random Forest o Gradient Boosting.



DECISION TREES FOR REGRESSION: PREDICTING A NUMBER

Regression with Decision Trees



Clasificación : Logística
Regresión : Números

podría querer predecir el peso de los animales con base a unas característica (lineal)

Using XGBoost

Classification

```
→ from xgboost import XGBClassifier  
→ model = XGBClassifier()  
→ model.fit(X_train, y_train)  
→ y_pred = model.predict(X_test)
```

Regression

```
from xgboost import XGBRegressor  
  
model = XGBRegressor()  
  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)
```

Árboles de Decisión vs Redes Neuronales

Árboles de Decisión y Conjuntos de Árboles

- Funcionan bien con datos tabulares (estructurados).
- No recomendados para datos no estructurados (imágenes, audio, texto).
- Rápidos.
- Los árboles de decisión pequeños pueden ser interpretables por humanos.

Redes Neuronales

- Funcionan bien con todo tipo de datos, incluyendo datos tabulares (estructurados) y no estructurados.
- Pueden ser más lentas que un árbol de decisión.
- Funcionan con aprendizaje por transferencia.
- Al construir un sistema de múltiples modelos trabajando juntos, podría ser más fácil encadenar múltiples redes neuronales.

A Comparative Analysis of XGBoost

^aCollege of Science

^bPázmány P

^bEscuela Polit

Why XGBoost models are derivative-free?



Moamen Elabd · [Follow](#)

5 min read · Feb 18, 2022

XGBoost is a scalable ensemble technique based on gradient boosting that has demonstrated to be a reliable and efficient machine learning challenge solver. This work proposes a practical analysis of how this novel technique works in terms of training speed, generalization performance and parameter setup. In addition, a comprehensive comparison between XGBoost, random forests and gradient boosting has been performed using

Binarización de Variables Categóricas:

1. **Variables Categóricas:** Son variables que toman un valor de un conjunto finito de categorías o niveles, como "rojo", "verde", "azul" para colores, o "bajo", "medio", "alto" para niveles.
2. **Binarización:** El proceso de binarización consiste en convertir estas variables categóricas en variables binarias (0 o 1). Esto se hace para que los modelos de aprendizaje automático, que generalmente trabajan con datos numéricos, puedan procesar estas variables.
3. **Métodos de Binarización:**
 - **One-Hot Encoding:** Es el método más común, donde se crea una nueva columna por cada categoría, y se coloca un 1 en la columna correspondiente a la categoría presente en esa fila, y 0 en las demás.
 - **Label Encoding:** Asigna un número entero a cada categoría. No es realmente una binarización, pero se utiliza en contextos similares.


Ejemplo de One-Hot Encoding:

- **Variable Original:** Color (Rojo, Verde, Azul)
- **Variables Binarizadas:**
 - Rojo: [1, 0, 0]
 - Verde: [0, 1, 0]
 - Azul: [0, 0, 1]

✓ 4. Building the Models

4.1 Decision Tree

Significa el mínimo número de ejemplos requerido para poder partir o generar un nodo, evito el overf



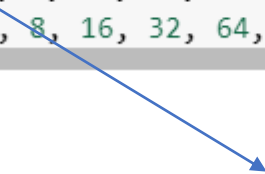
The hyperparameters we will use and investigate here are:

- `min_samples_split`: The minimum number of samples required to split an internal node.
 - Choosing a higher `min_samples_split` can reduce the number of splits and may help to reduce overfitting.
- `max_depth`: The maximum depth of the tree.
 - Choosing a lower `max_depth` can reduce the number of splits and may help to reduce overfitting.

```
[ ] min_samples_split_list = [2,10, 30, 50, 100, 200, 300, 700] ## If the number is an integer, then it is  
max_depth_list = [1,2, 3, 4, 8, 16, 32, 64, None] # None means that there is no depth limit.
```



Ensaye con /



2 Elementos , 10 pare en 10 , no puede hacer mas divisiones

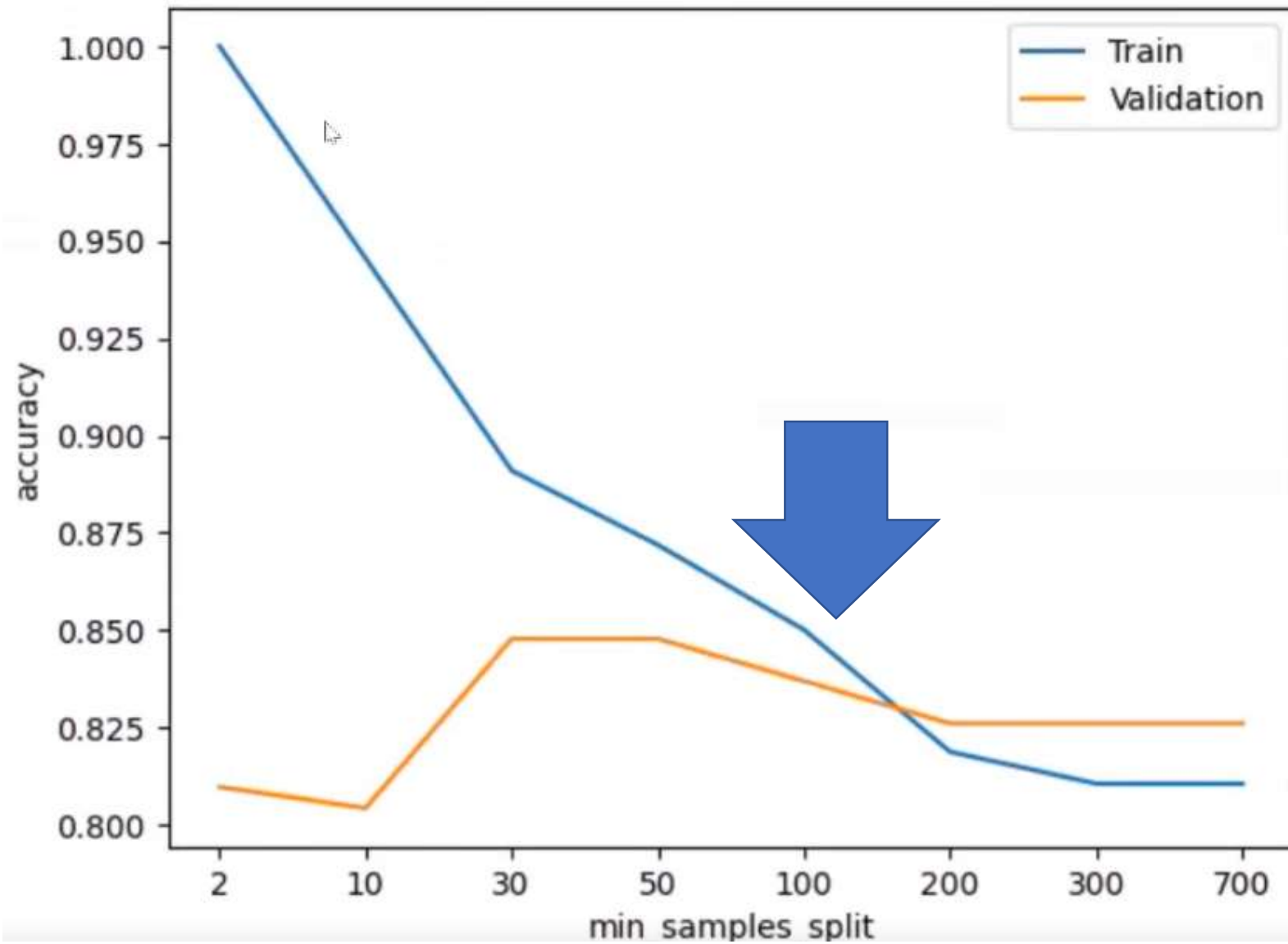
por cada elementos en esta lista haga un modelo 8 va a traer un clasificador de árbol de decisión

```
accuracy_list_train = []
accuracy_list_val = []
for min_samples_split in min_samples_split_list:
    model = DecisionTreeClassifier(min_samples_split = min_samples_split,
                                   random_state=RANDOM_STATE).fit(X_train, y_train)
    predictions_train = model.predict(X_train) #this are the predicted values for the train dataset
    prediction_accuracy_train = accuracy_score(predictions_train, y_train)
    accuracy_list_train.append(prediction_accuracy_train)
    accuracy_list_val.append(accuracy_score(predictions_val, y_val))

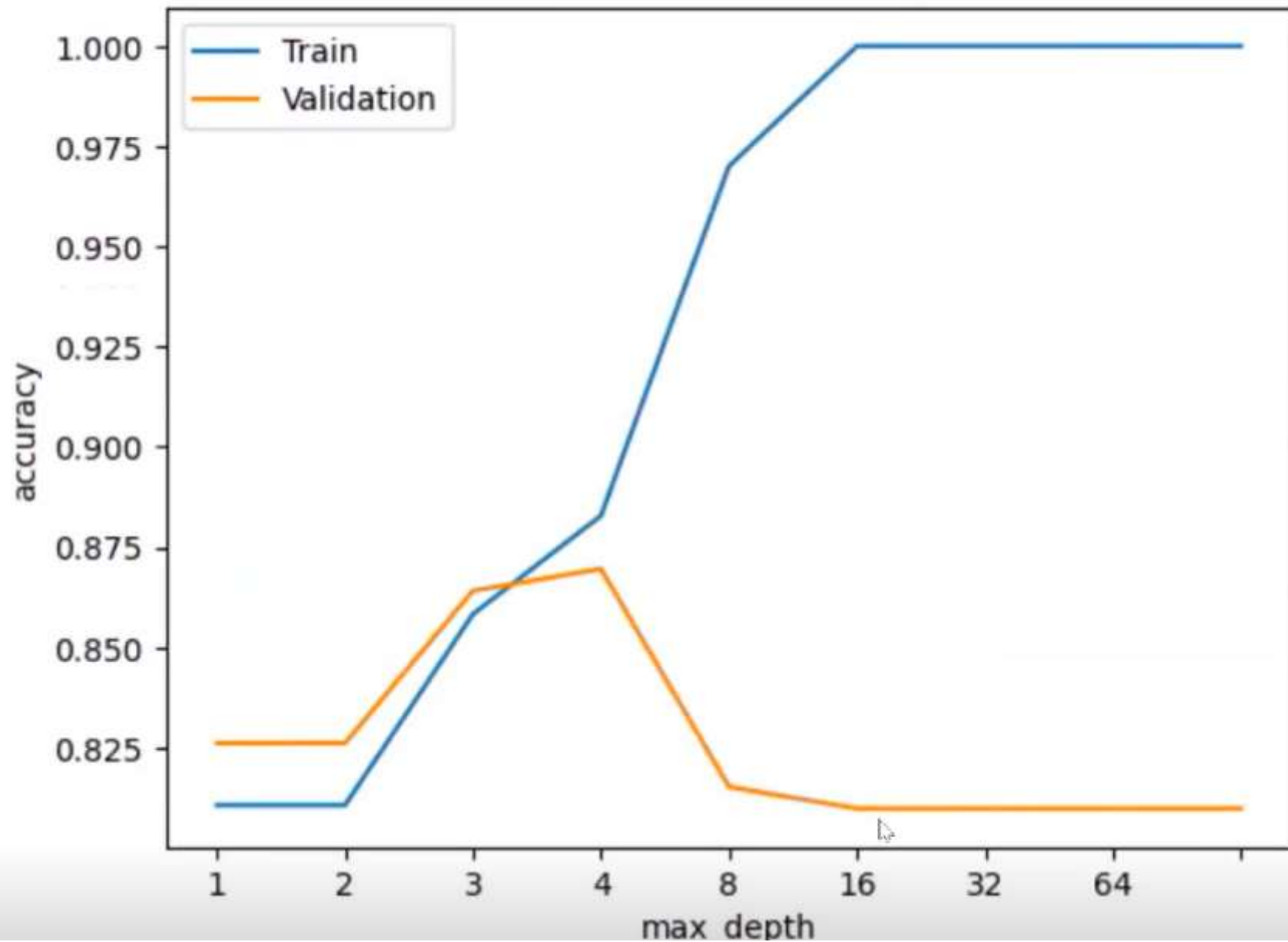
plt.title('Train x Validation metrics')
plt.xlabel('min_samples_split')
plt.ylabel('accuracy')
plt.xticks(ticks = range(len(min_samples_split_list)), labels=min_samples_split_list)
plt.plot(accuracy_list_train)
plt.plot(accuracy_list_val)
plt.legend(['Train', 'Validation'])
```

Entrene con x y y

Train x Validation metrics



Train x Validation metrics

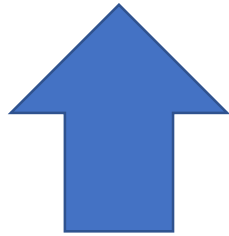


✓ 4.2 Random Forest

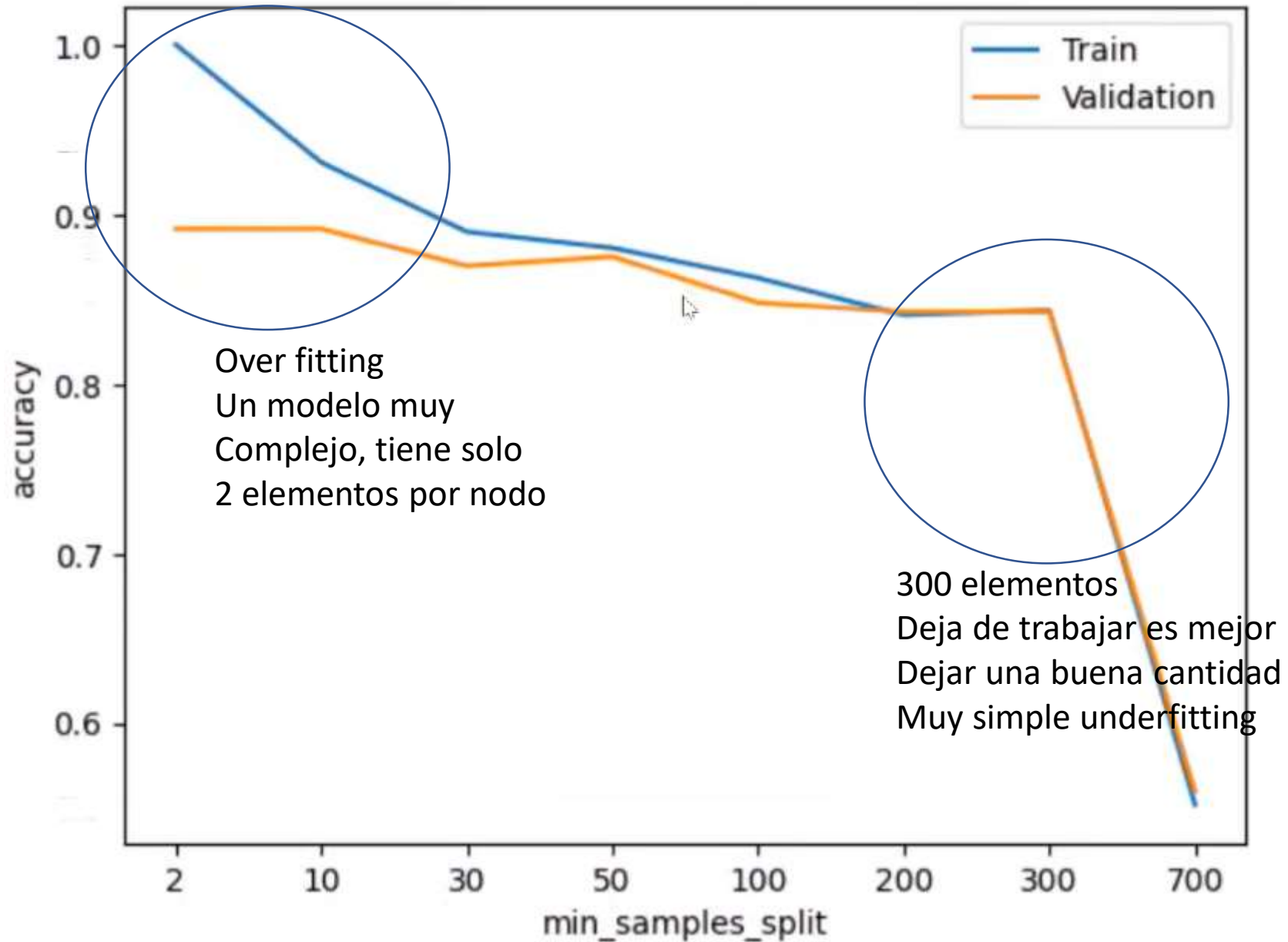
the Random Forest algorithm also, using the Scikit-learn implementation.

- All of the hyperparameters found in the decision tree model will also exist in this algorithm, since a random forest is an ensemble of many Decision Trees.
- One additional hyperparameter for Random Forest is called `n_estimators` which is the number of Decision Trees that make up the Random Forest.

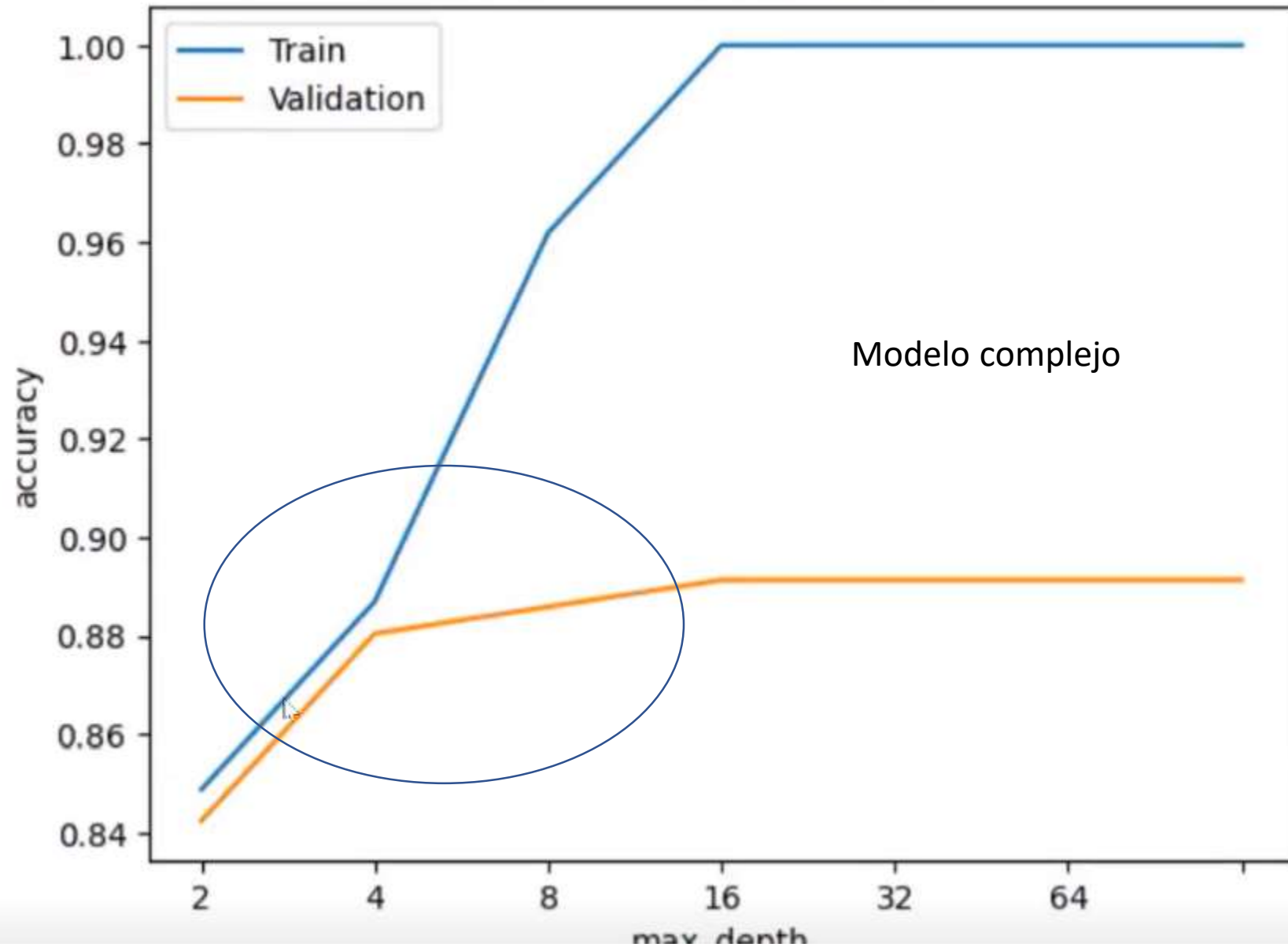
```
min_samples_split_list = [2,10, 30, 50, 100, 200, 300, 700] ## If the number is an integer, then it is the actual quantity of samples
..... ## If it is a float, then it is the percentage of the dataset
max_depth_list = [2, 4, 8, 16, 32, 64, None]
n_estimators_list = [10,50,100,500]
```



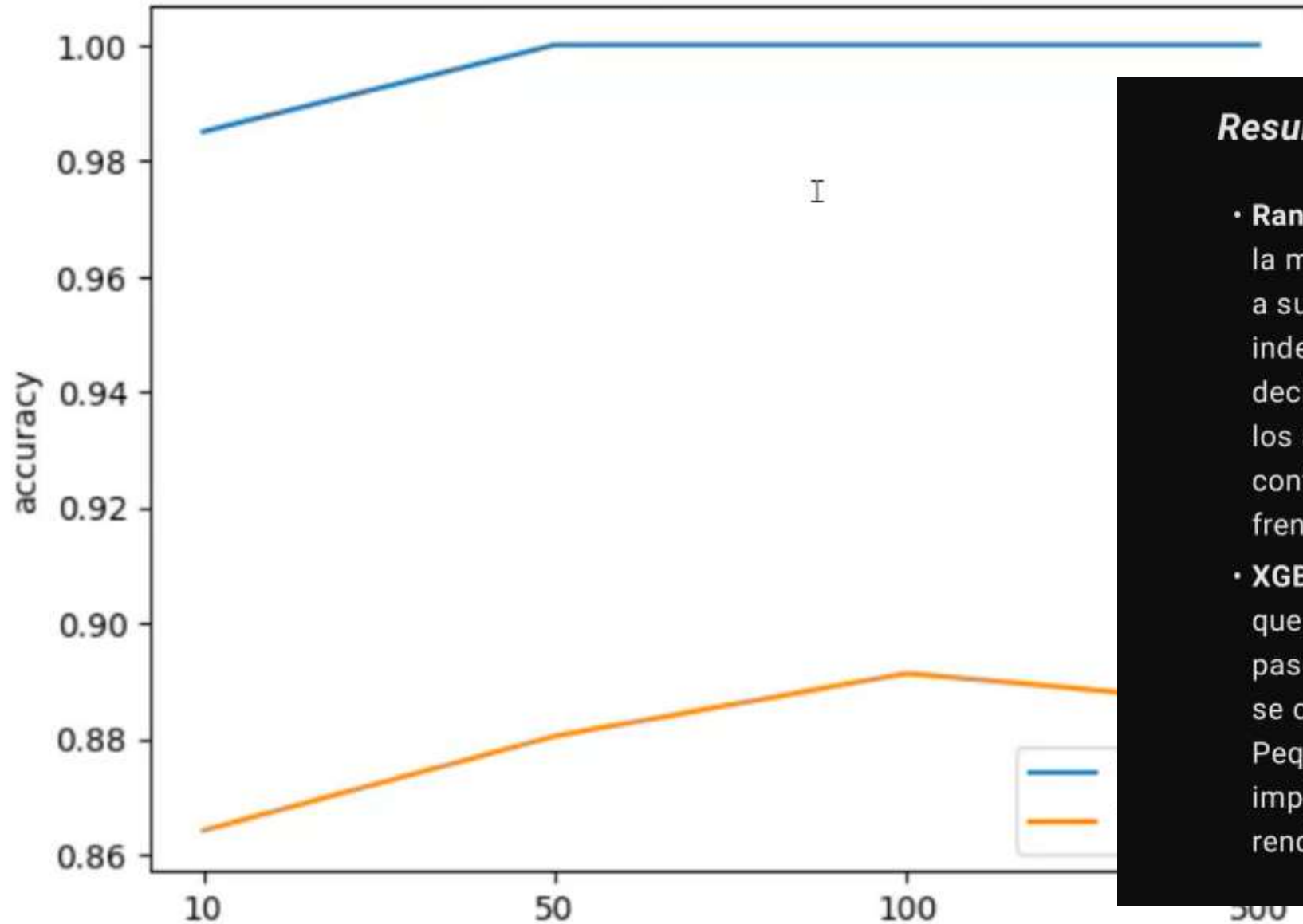
Train x Validation metrics



Train x Validation metrics



Train x Validation metrics



Resumen:

- **Random Forest** es menos sensible a la modificación de parámetros debido a su naturaleza de ensamble y la independencia de los árboles de decisión. Los errores individuales de los árboles se promedian, lo que contribuye a la estabilidad del modelo frente a cambios de parámetros.
- **XGBoost**, al ser un modelo secuencial que busca corregir errores en cada paso, es mucho más sensible a cómo se configuran los parámetros. Pequeños ajustes pueden tener un impacto considerable en el rendimiento, para bien o para mal.

✓ 0 s [45] random_forest_model = RandomForestClassifier(n_estimators = 100,
max_depth = 16,
min_samples_split = 10).fit(X_train,y_train)

✓ 0 s ▶ print(f"Metrics train:\n\tAccuracy score: {accuracy_score(random_forest_model.predict(X_train),

↔ Metrics train:
Accuracy score: 0.9401
Metrics test:
Accuracy score: 0.9076

✓ 4.3 XGBoost

.3 XGBoost

El siguiente modelo es el de Gradient Boosting, llamado XGBoost. Los métodos de boosting entrenan varios árboles, pero en lugar de ser no correlacionados entre sí, ahora los árboles se ajustan uno tras otro con el fin de minimizar el error.

El modelo tiene los mismos parámetros que un árbol de decisión, más la tasa de aprendizaje.

La tasa de aprendizaje es el tamaño del paso en el método de descenso de gradiente que XGBoost utiliza internamente para minimizar el error en cada paso de entrenamiento. Una característica interesante del XGBoost es que, durante el ajuste, puede aceptar un conjunto de datos de evaluación con la forma (X_val, y_val).

En cada iteración, mide el costo (o métrica de evaluación) en los conjuntos de datos de evaluación. Una vez que el costo (o métrica) deja de disminuir durante un número determinado de rondas (llamadas `early_stopping_rounds`), el entrenamiento se detendrá. Más iteraciones conducen a más estimadores, y más estimadores pueden resultar en sobreajuste. Al detener el entrenamiento una vez que la métrica de validación deja de mejorar, podemos limitar el número de estimadores creados y reducir el sobreajuste. Primero, definamos un subconjunto de nuestro conjunto de entrenamiento (no debemos usar el conjunto de prueba aquí).

```
[ ] n = int(len(X_train)*0.8) ##let's use 80% to train and 20% to eval
```

```
[ ] X_train_fit, X_train_eval, y_train_fit, y_train_eval = X_train[:n], X_train[n:], y_train[:n], y_train[n:]
```

```
[ ] pip install --upgrade xgboost
```

```
[ ] xgb_model = XGBClassifier(n_estimators=500, learning_rate=0.1, verbosity=1, random_state=RANDOM_STATE)  
xgb_model.fit(X_train_fit, y_train_fit, eval_set=[(X_train_eval, y_train_eval)])
```

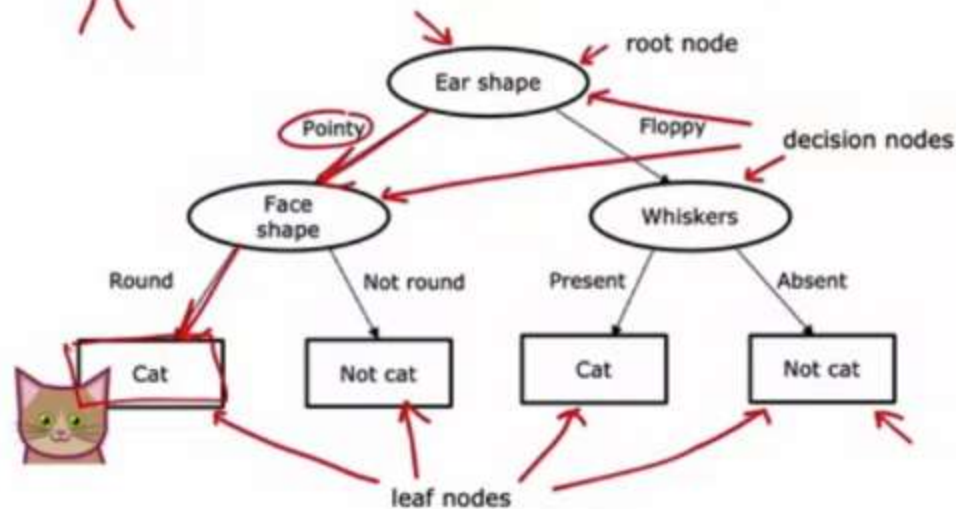
```
[486] validation_0-logloss:0.64316  
[487] validation_0-logloss:0.64327  
[488] validation_0-logloss:0.64358  
[489] validation_0-logloss:0.64351  
[490] validation_0-logloss:0.64386  
[491] validation_0-logloss:0.64351  
[492] validation_0-logloss:0.64308  
[493] validation_0-logloss:0.64317  
[494] validation_0-logloss:0.64331  
[495] validation_0-logloss:0.64370  
[496] validation_0-logloss:0.64362  
[497] validation_0-logloss:0.64373  
[498] validation_0-logloss:0.64344  
[499] validation_0-logloss:0.64379
```

EJEMPLO PRACTICO

1.



Decision Tree



New test example

Ear shape: Pointy
Face shape: Round
Whiskers: Present

1 / 1 punt

Basándose en el árbol de decisión mostrado en la conferencia, si un animal tiene orejas caídas, una forma de cara redonda y tiene bigotes, ¿predice el modelo que es un gato o que no es un gato?

- ☐ No es un gato
- ☒ cat

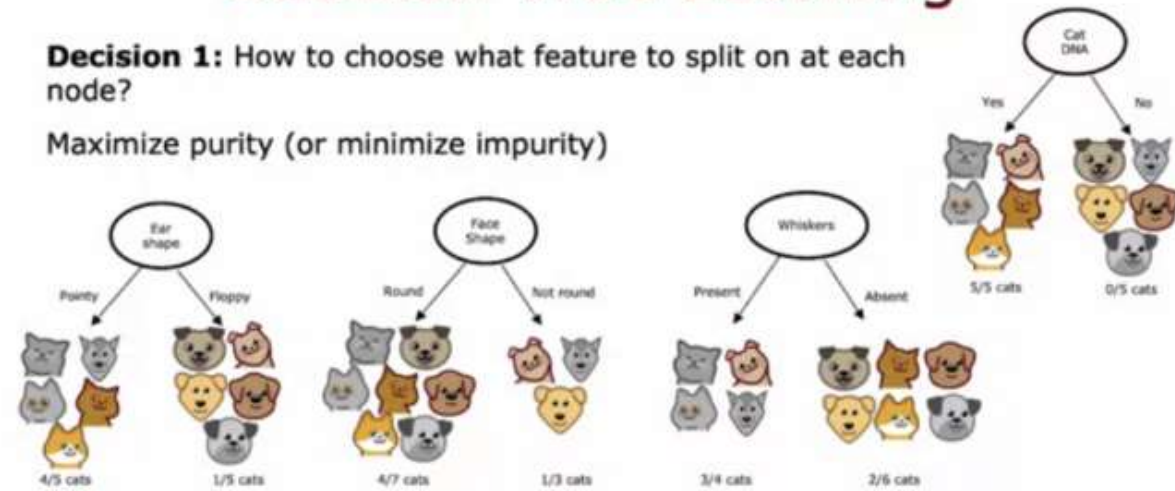
✓ Correcto

Correcto. Si sigue las orejas caídas hacia la derecha y, a continuación, desde el nodo de decisión de los bigotes, va hacia la izquierda porque los bigotes están presentes, llegará a un nodo hoja para "gato", por lo que el modelo predeciría que se trata de un gato.

Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?

Maximize purity (or minimize impurity)



Tome un aprendizaje de árbol de decisión para clasificar entre correo electrónico spam y no spam. Hay 20 ejemplos de entrenamiento en la nota raíz, que comprenden 10 correos electrónicos de spam y 10 de no spam. Si el algoritmo puede elegir entre cuatro características, que dan lugar a cuatro divisiones correspondientes, ¿cuál elegiría (es decir, cuál tiene mayor pureza)?

- ☐ División izquierda: 5 de 10 correos electrónicos son spam. División derecha: 5 de 10 correos electrónicos son spam.
- ☒ División izquierda: 10 de 10 correos electrónicos son spam. División derecha: 0 de 10 correos electrónicos son spam.
- ☐ División izquierda: 2 de 2 correos electrónicos son spam. División derecha: 8 de 18 correos electrónicos son spam.
- ☐ División izquierda: 7 de 8 correos electrónicos son spam. División derecha: 3 de 12 correos electrónicos son spam.

✓ Correcto
¡Sí!