

# TICS-411 Minería de Datos

Clase 8: Evaluación de Modelos

Alfonso Tobar-Arancibia

alfonso.tobar.a@edu.uai.cl

# Evaluación de Modelos Supervisados

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.
- ¿Qué pasa si ahora les doy la respuesta?

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.
- ¿Qué pasa si ahora les doy la respuesta?
- ¿Qué pasa si te doy más ejercicios?

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.
- ¿Qué pasa si ahora les doy la respuesta?
- ¿Qué pasa si te doy más ejercicios?
- ¿Qué pasa luego de que haces muchos ejercicios?

# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.
- ¿Qué pasa si ahora les doy la respuesta?
- ¿Qué pasa si te doy más ejercicios?
- ¿Qué pasa luego de que haces muchos ejercicios?



Voy **aprendiendo mejor la tarea** de calcular el coeficiente de Silueta. Lo mismo pasa con los modelos.



# Intuición

Supongamos que tengo que estudiar para la prueba de **Minería de Datos** y tengo que aprender a calcular el Coeficiente de Silueta.

- Qué pasa si sólo les entrego una pregunta para estudiar y no tiene respuesta.
- ¿Qué pasa si ahora les doy la respuesta?
- ¿Qué pasa si te doy más ejercicios?
- ¿Qué pasa luego de que haces muchos ejercicios?

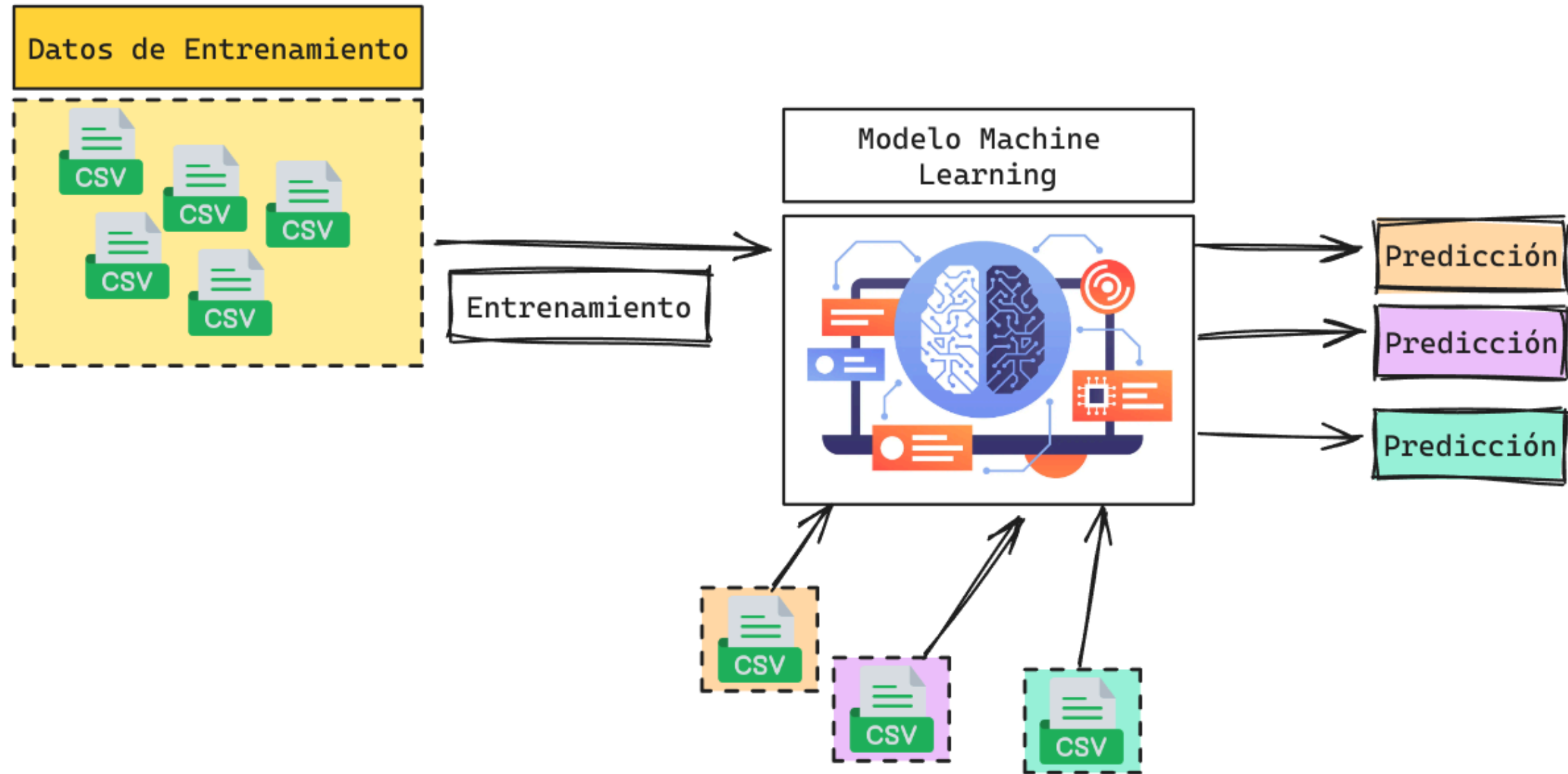


Voy **aprendiendo mejor la tarea** de calcular el coeficiente de Silueta. Lo mismo pasa con los modelos.

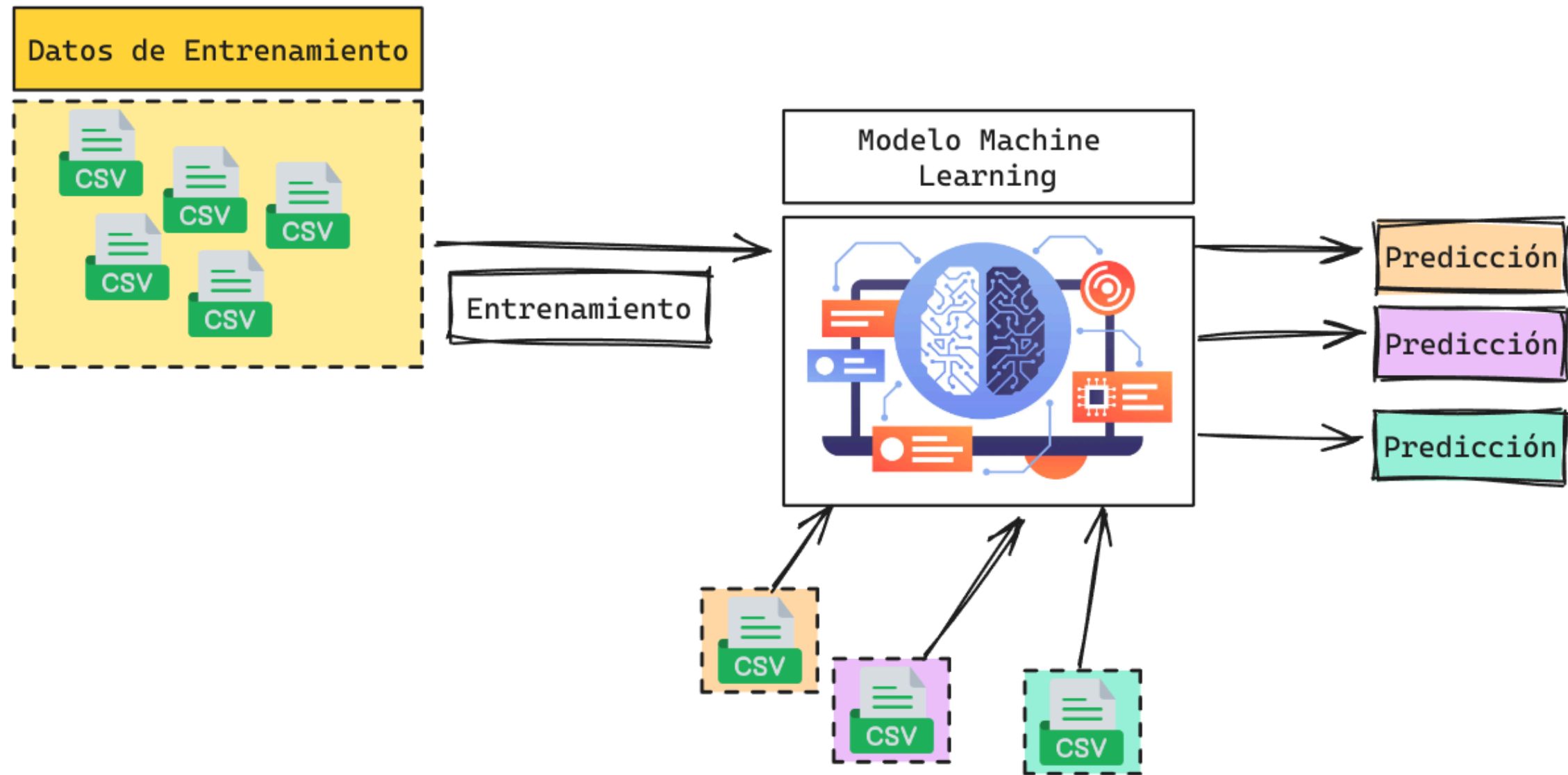


Pero no puedo medir qué tan bien aprendiste en los ejercicios que yo ya entregué para practicar. **Tengo que hacer una prueba que tú no hayas visto, para ver si realmente aprendiste.**

# Uso de un Modelo



# Uso de un Modelo



¿Cómo saber que el modelo está funcionando como esperamos?

# Evaluación de un Modelo

# Métricas

El Rendimiento de un Modelo de Clasificación permite evaluar el **error** asociado al proceso de predicción.

## Clase Positiva

Corresponde a la clase/evento de interés. Ej: Tiene cancer, va a pagar su deuda, es un gato. Normalmente se denota como la **Clase 1**.

## Clase Negativa

Corresponde a la clase/evento contrario al de interés. Ej: No tiene cancer, no va a pagar su deuda, no es un gato. Normalmente se denota como la **Clase 0**.

💡 **Scikit-Learn** usa la siguiente convención:

- Si se llama **\*\_score** un mayor puntaje es mejor.
- Si se llama **\*\_error** o **\*\_loss** un mejor puntaje es mejor.

# Métricas: Matriz de Confusión

La Matriz de Confusión ordena los valores correctamente predichos y también los distintos errores que el modelo puede cometer.

		Predichos	
		0	1
Reales	0	TN	FP
	1	FN	TP

## TP (Verdaderos Positivos)

Corresponde a valores reales de la clase 1 que fueron correctamente predichos como clase 1.

## TN (Verdaderos Negativos)

Corresponde a valores reales de la clase 0 que fueron correctamente predichos como clase 0.

## FP (Falsos Positivos)

Corresponde a valores reales de la clase 0 que fueron incorrectamente predichos como clase 1.

## FN (Falsos Negativos)

Corresponde a valores reales de la clase 1 que fueron incorrectamente predichos como clase 0.

# Métricas: A partir de la Matriz de Confusión

Accuracy

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Recall

$$\frac{TP}{TP + FN}$$

Precision

$$\frac{TP}{TP + FP}$$

F1-Score

$$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$



- **Accuracy** es probablemente la métrica más sencilla y más utilizada.
- **Precision** y **Recall** ponderarán distintos errores (FP y FN respectivamente) con mayor severidad. Ambas métricas son Antagonistas.
- **F1-Score** corresponde a la media armónica del **Precision** y **Recall**, y tiende a ponderar los errores de manera más balanceada.

## ¿Cuándo utilizar cada tipo de error?

# Curva ROC

La curva ROC fue desarrollada en 1950 para analizar señales ruidosas. La curva ROC permite al operador contrapesar la tasa de verdaderos positivos (Eje  $y$ ) versus los falsos positivos (Eje  $x$ ).

El área bajo la curva representa la **calidad del modelo**. Una manera de interpretarla es como la probabilidad de que una predicción de la clase positiva tenga mayor probabilidad que una de clase negativa. En otras palabras, **mide que las probabilidades se encuentren correctamente ordenadas**. Por lo tanto varía entre 0.5 y 1.



# Implementación en Python

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
2
3 accuracy_score(y_true, y_pred)
4 precision_score(y_true, y_pred)
5 recall_score(y_true, y_pred)
6 f1_score(y_true, y_pred)
7 roc_auc_score(y_true, y_proba)
```

- **y\_true**: Corresponde a las etiquetas reales del Dataset.
- **y\_pred**: Corresponde a las predicciones realizadas por el modelo.
- **y\_proba**: Corresponden a las probabilidades predichas por el modelo (si es que el modelo lo permite).

# Implementación en Python

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
2
3 accuracy_score(y_true, y_pred)
4 precision_score(y_true, y_pred)
5 recall_score(y_true, y_pred)
6 f1_score(y_true, y_pred)
7 roc_auc_score(y_true, y_proba)
```

- **y\_true**: Corresponde a las etiquetas reales del Dataset.
- **y\_pred**: Corresponde a las predicciones realizadas por el modelo.
- **y\_proba**: Corresponden a las probabilidades predichas por el modelo (si es que el modelo lo permite).

# Implementación en Python

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
2
3 accuracy_score(y_true, y_pred)
4 precision_score(y_true, y_pred)
5 recall_score(y_true, y_pred)
6 f1_score(y_true, y_pred)
7 roc_auc_score(y_true, y_proba)
```

- **y\_true**: Corresponde a las etiquetas reales del Dataset.
- **y\_pred**: Corresponde a las predicciones realizadas por el modelo.
- **y\_proba**: Corresponden a las probabilidades predichas por el modelo (si es que el modelo lo permite).

# Implementación en Python

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
2
3 accuracy_score(y_true, y_pred)
4 precision_score(y_true, y_pred)
5 recall_score(y_true, y_pred)
6 f1_score(y_true, y_pred)
7 roc_auc_score(y_true, y_proba)
```

- **y\_true**: Corresponde a las etiquetas reales del Dataset.
- **y\_pred**: Corresponde a las predicciones realizadas por el modelo.
- **y\_proba**: Corresponden a las probabilidades predichas por el modelo (si es que el modelo lo permite).

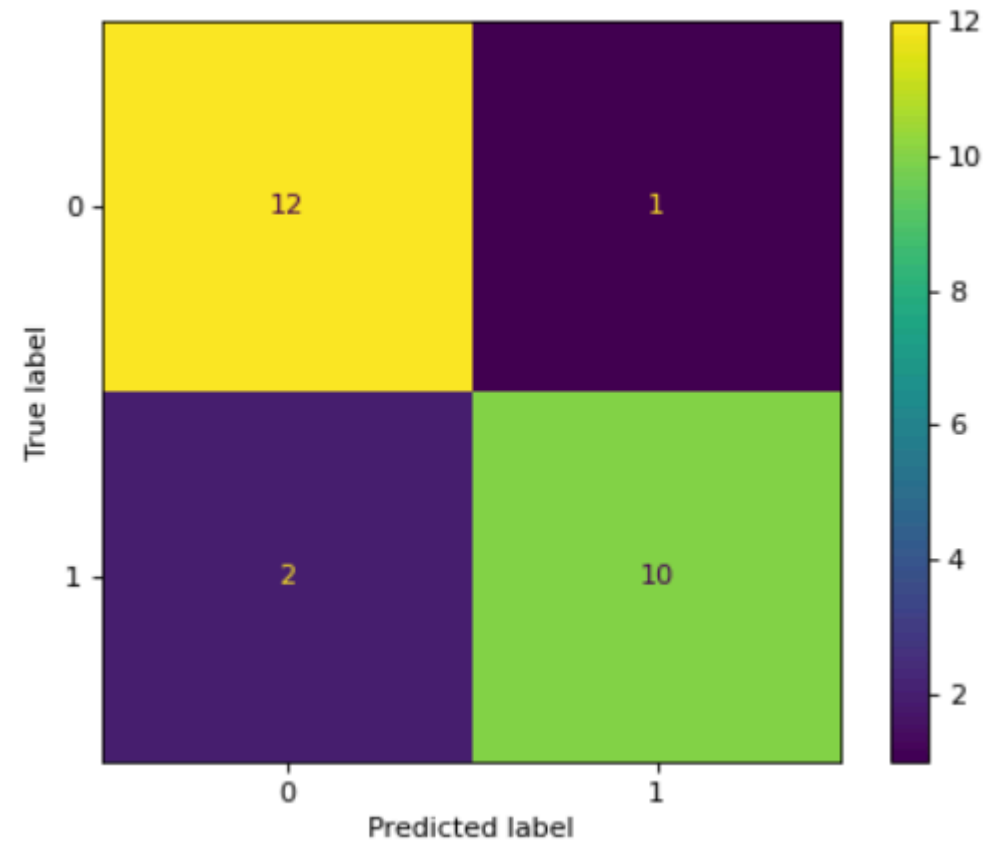
# Implementación en Python

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
2
3 accuracy_score(y_true, y_pred)
4 precision_score(y_true, y_pred)
5 recall_score(y_true, y_pred)
6 f1_score(y_true, y_pred)
7 roc_auc_score(y_true, y_proba)
```

- **y\_true**: Corresponde a las etiquetas reales del Dataset.
- **y\_pred**: Corresponde a las predicciones realizadas por el modelo.
- **y\_proba**: Corresponden a las probabilidades predichas por el modelo (si es que el modelo lo permite).

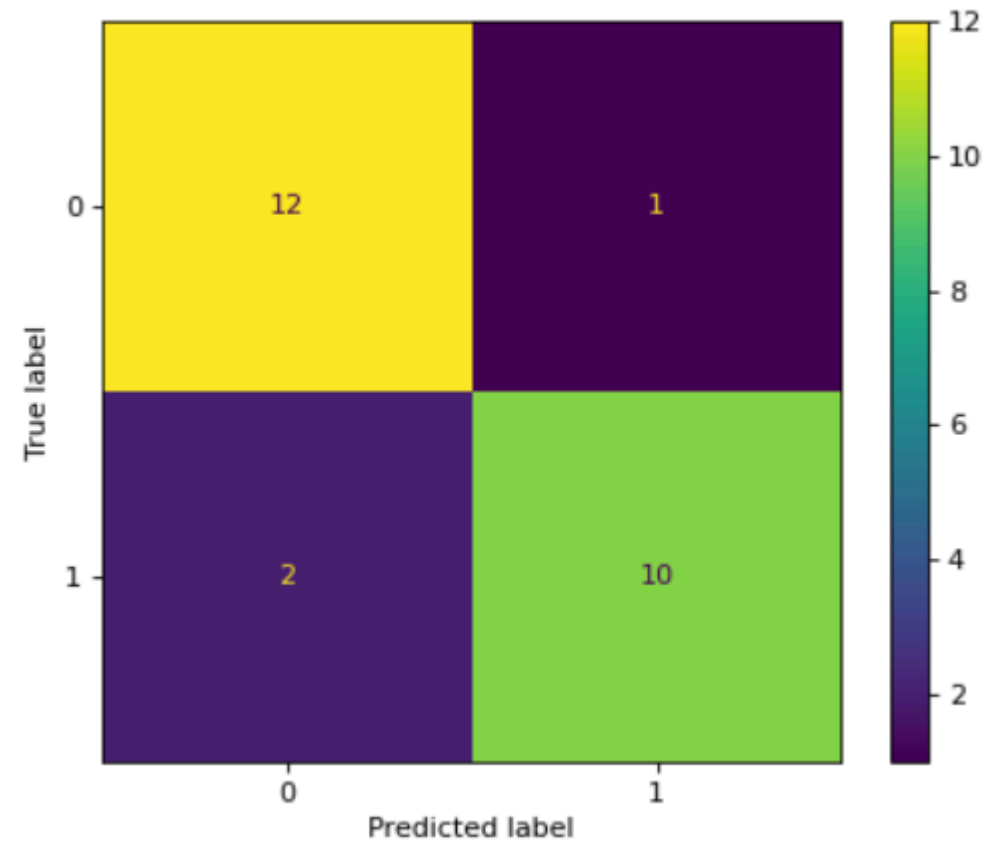
# Implementación en Python: Matriz de Confusión

```
1 from sklearn.metrics import ConfusionMatrixDisplay
2
3 ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```



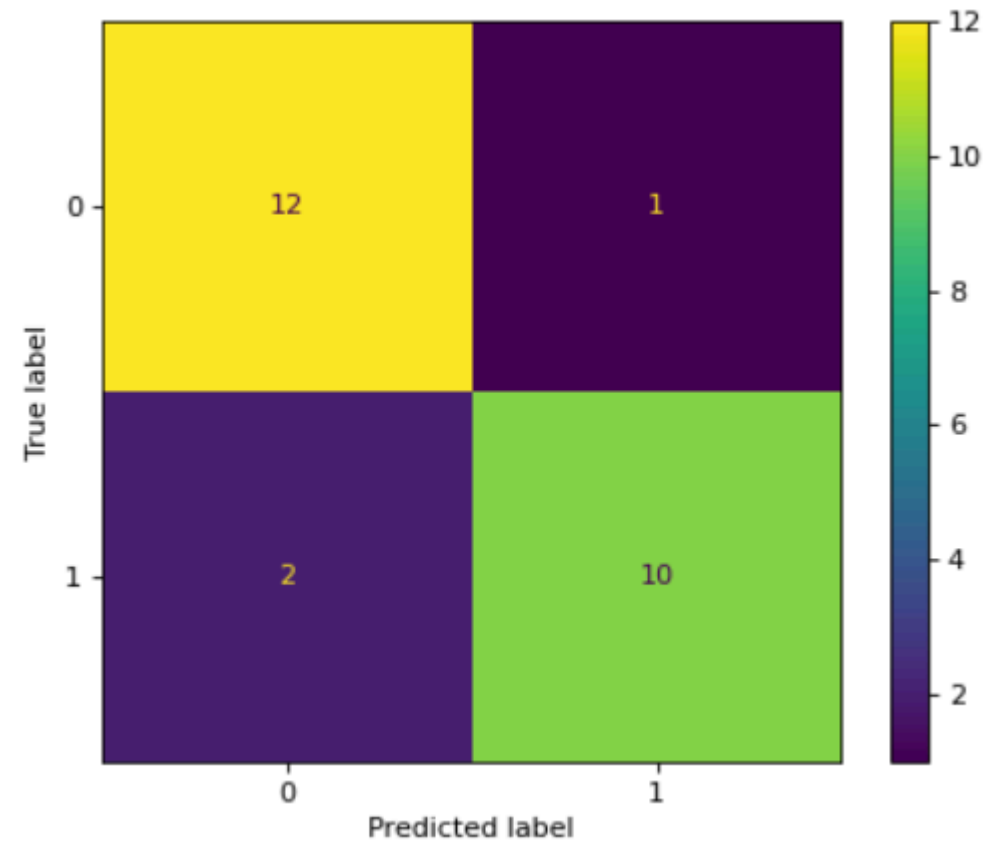
# Implementación en Python: Matriz de Confusión

```
1 from sklearn.metrics import ConfusionMatrixDisplay
2
3 ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```



# Implementación en Python: Matriz de Confusión

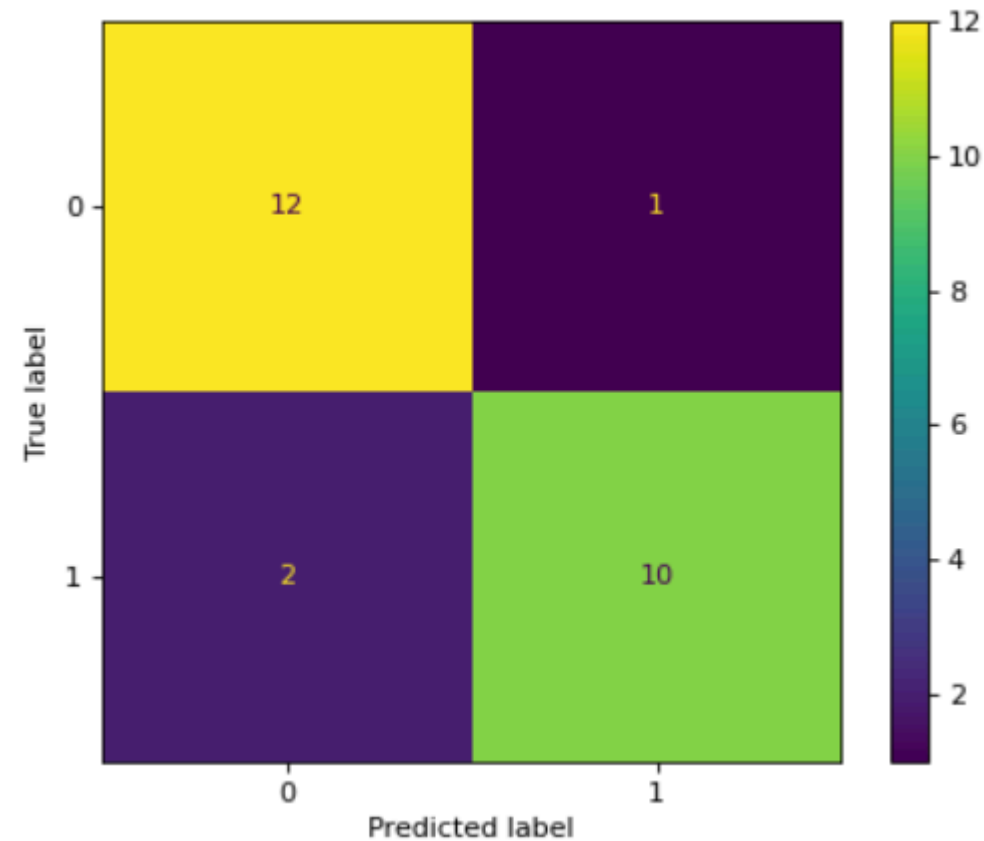
```
1 from sklearn.metrics import ConfusionMatrixDisplay
2
3 ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```





# Implementación en Python: Matriz de Confusión

```
1 from sklearn.metrics import ConfusionMatrixDisplay
2
3 ConfusionMatrixDisplay.from_predictions(y_true, y_pred)
```



# Implementación en Python: Curva ROC

```
1 from sklearn.metrics import RocCurveDisplay
2
3 RocCurveDisplay.from_predictions(y_true, y_proba)
```



# Implementación en Python: Curva ROC

```
1 from sklearn.metrics import RocCurveDisplay
2
3 RocCurveDisplay.from_predictions(y_true, y_proba)
```



# Implementación en Python: Curva ROC

```
1 from sklearn.metrics import RocCurveDisplay
2
3 RocCurveDisplay.from_predictions(y_true, y_proba)
```



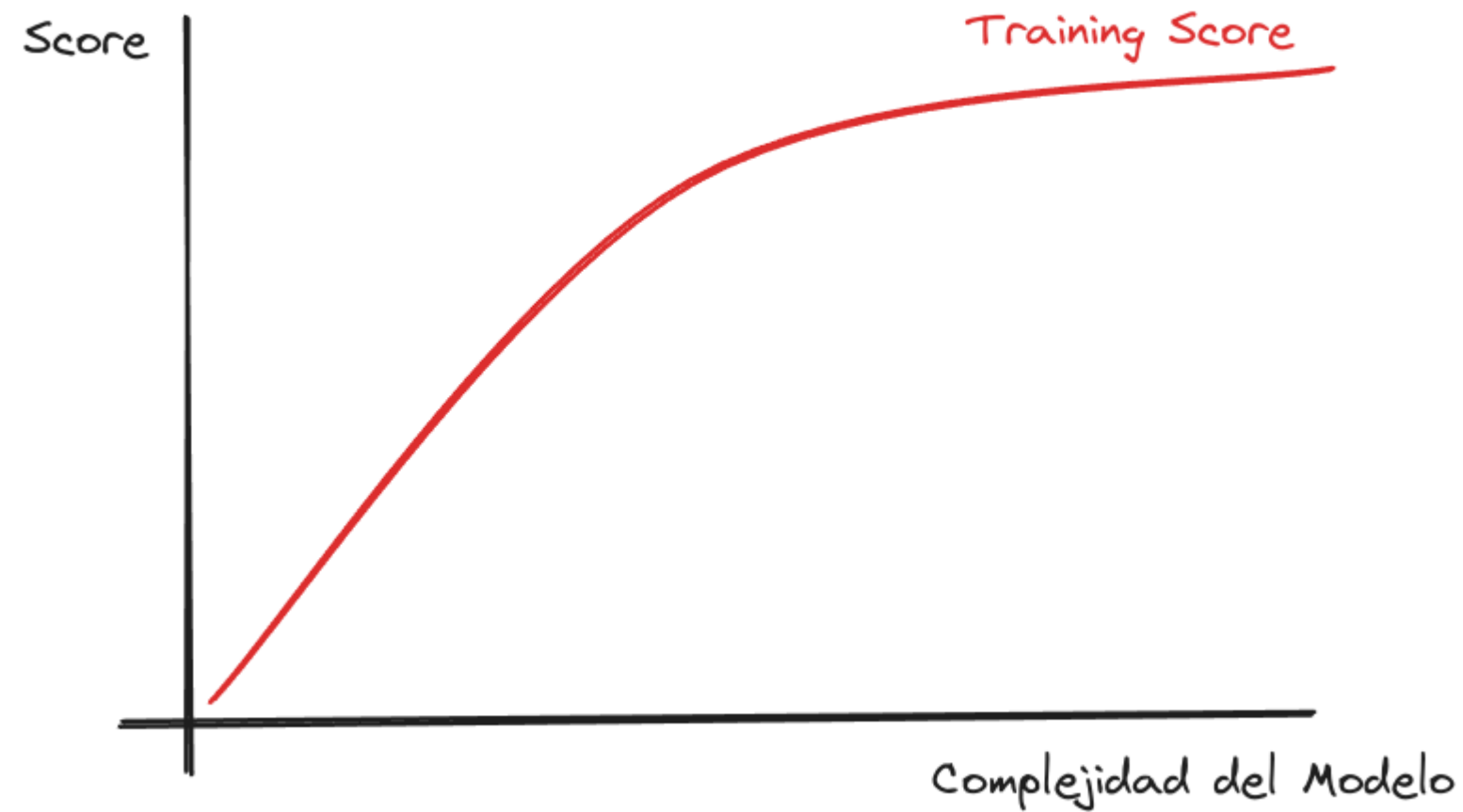
# Implementación en Python: Curva ROC

```
1 from sklearn.metrics import RocCurveDisplay
2
3 RocCurveDisplay.from_predictions(y_true, y_proba)
```



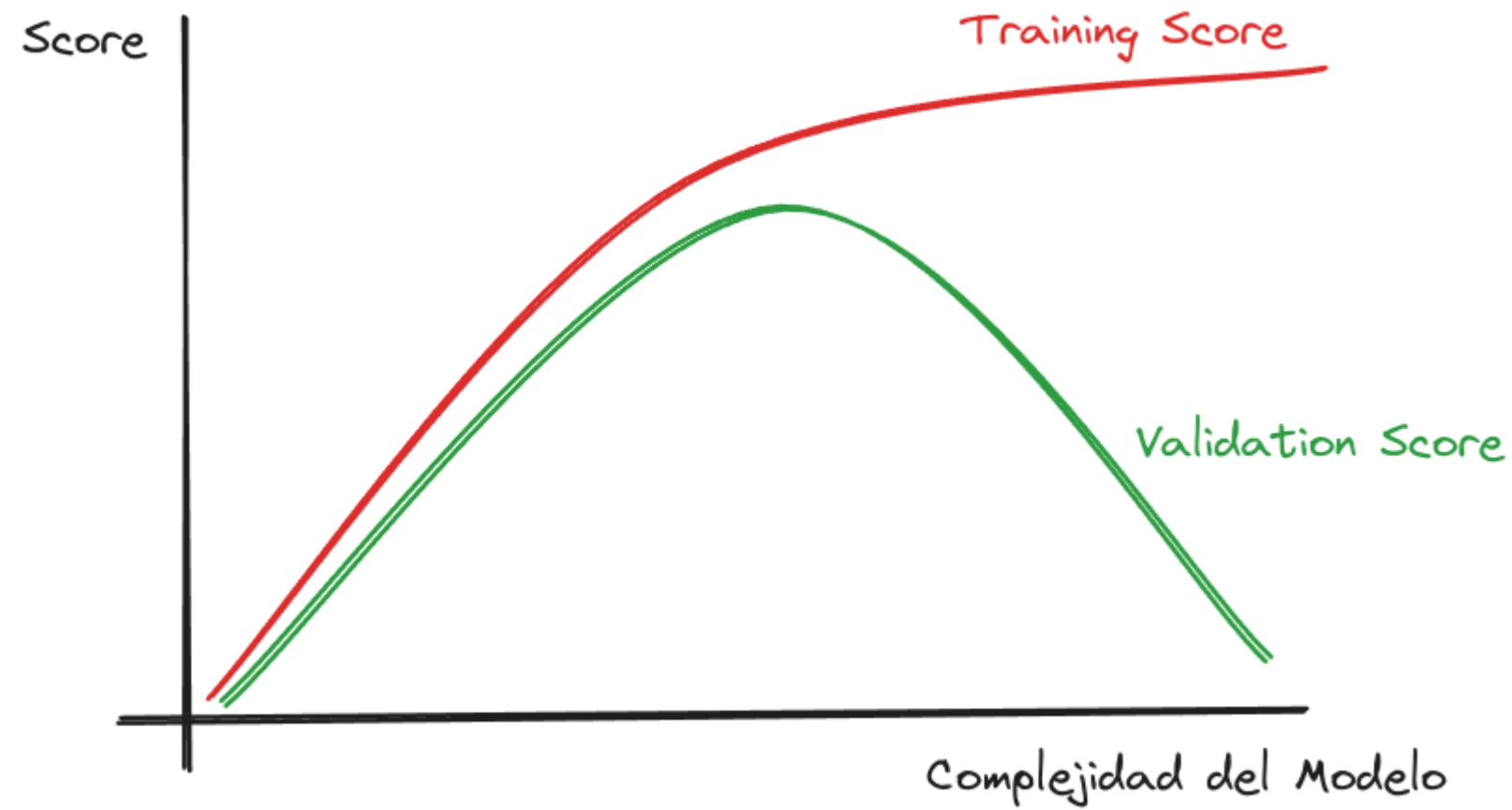
# Sobre el aprendizaje de un Modelo

# Curva de Aprendizaje: Training



 ¿Qué sería la Complejidad del Modelo?

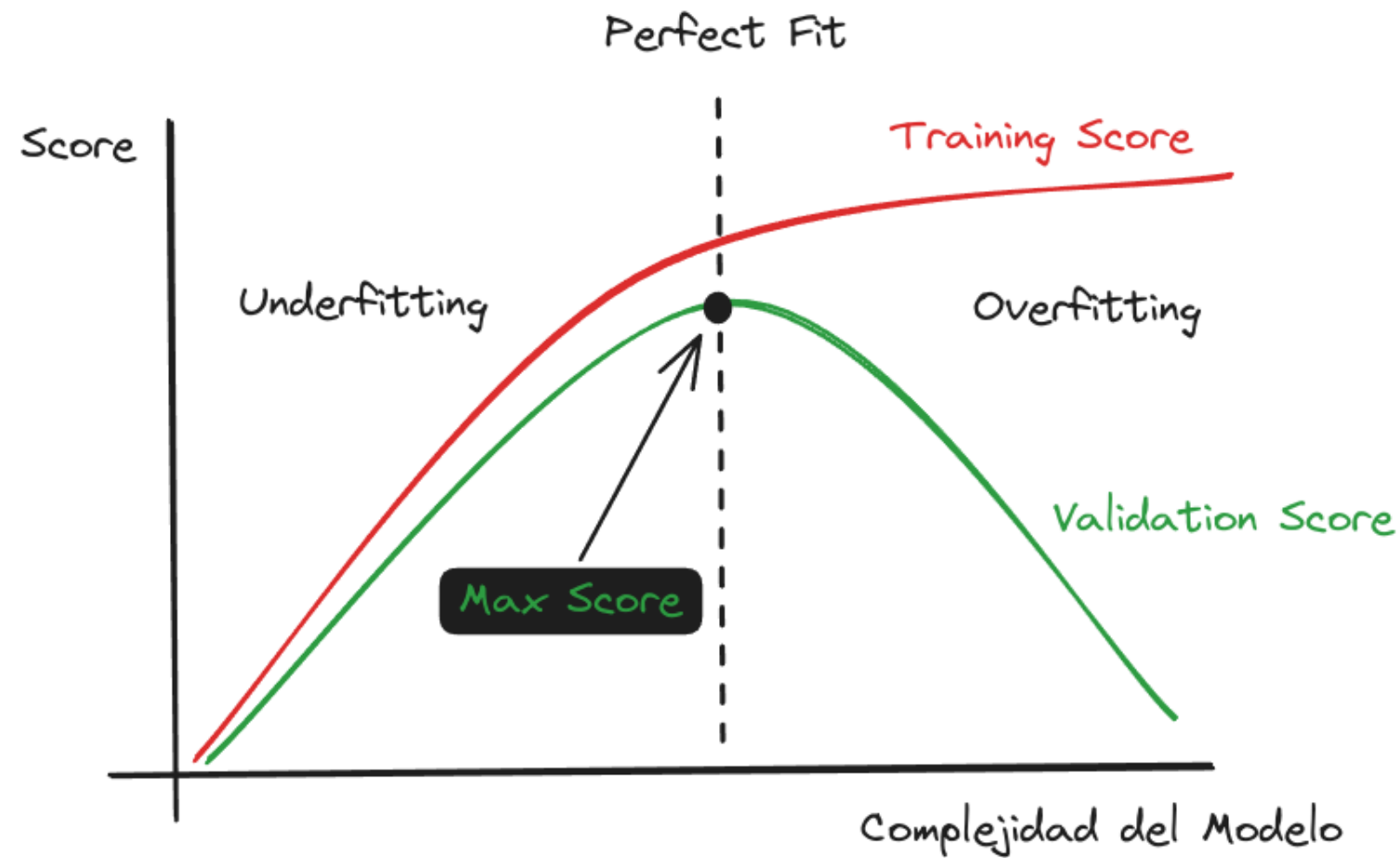
# Curva de Aprendizaje: Validación



¿Por qué el modelo pierde rendimiento cuando aumenta su Complejidad?



# Curva de Aprendizaje: Mejor Ajuste



## Overfitting:

Gran diferencia entre Training y Validation Score.



## Underfitting:

Poca diferencia entre Training y Validation Score, pero con ambos puntajes "relativamente bajos".



## Proper fitting:

Corresponde al mejor puntaje en el set de Validación. Donde también la distancia entre Train y Test es *poca*.

# Evaluación

La evaluación de modelos supervisados es fundamental. De no hacerlo de forma correcta podemos quedarnos con una idea muy equivocada del rendimiento del modelo.



Un modelo con un rendimiento incorrecto puede entregar predicciones completamente inútiles.

## Cross Validation (Validación Cruzada)

Se debe evaluar el rendimiento de un modelo en un **dataset diferente** al que fue entrenado. Esta es la única manera en la que se puede medir el poder de generalización.

## Generalización

Corresponde a la habilidad de un modelo de adaptarse apropiadamente a datos no vistos previamente.



Para esto se asume que todos los datos son i.i.d (**independent and identically distributed**). De no lograr esto, lograr buenos rendimientos es más difícil.

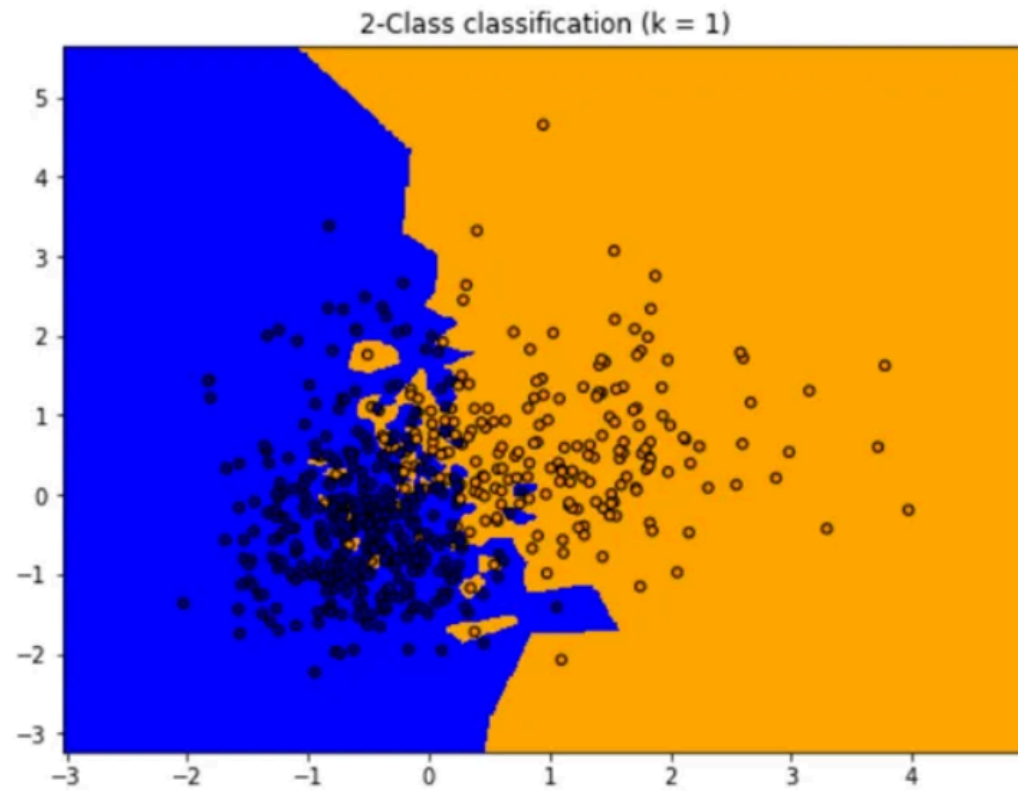
# Complejidad de un Modelo

¿Qué modelo es un mejor clasificador?

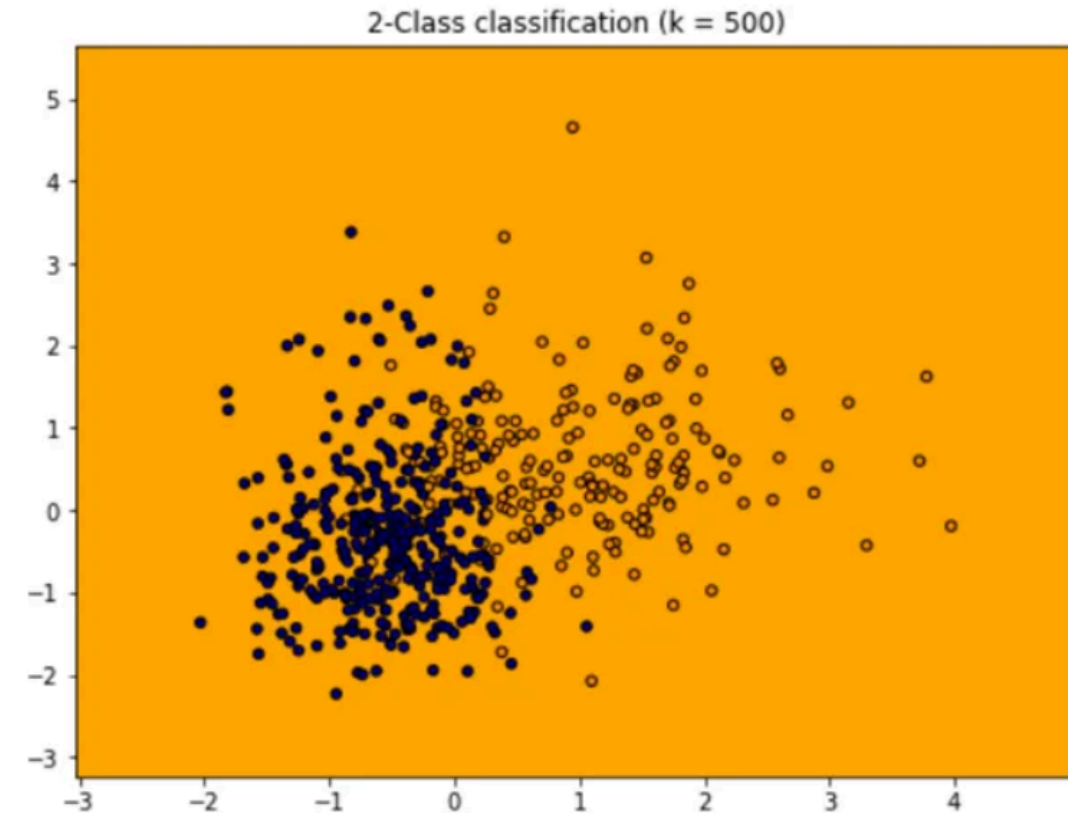


# Complejidad de un Modelo

## Overfitting

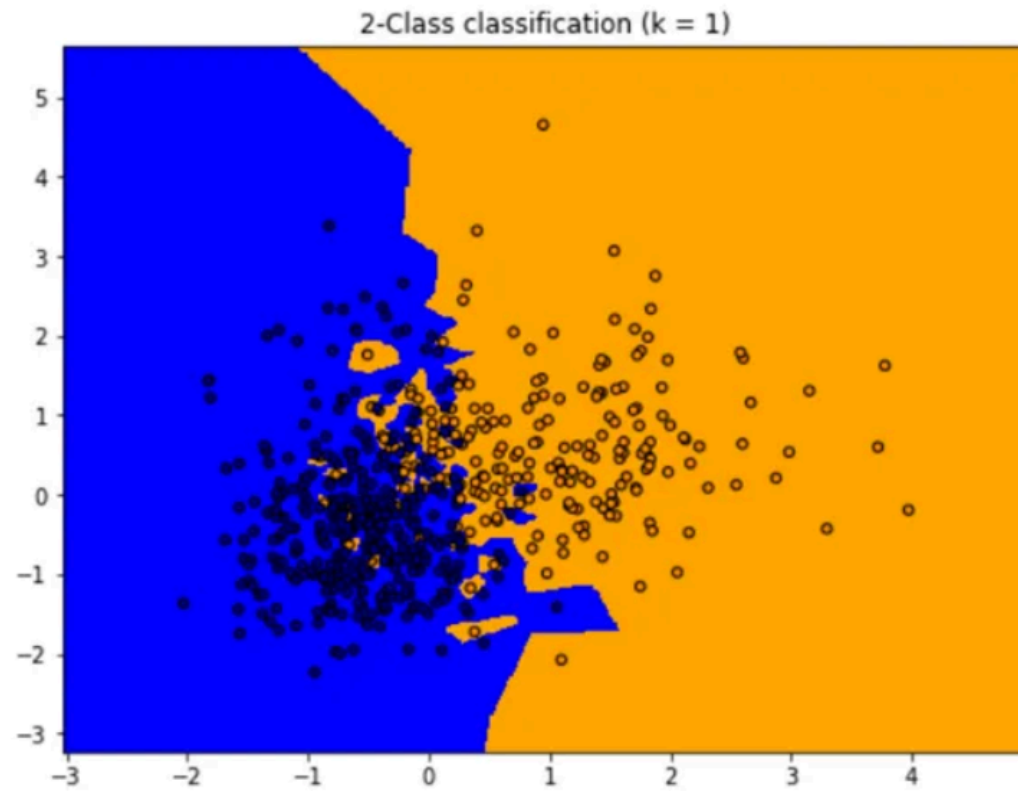


## Underfitting

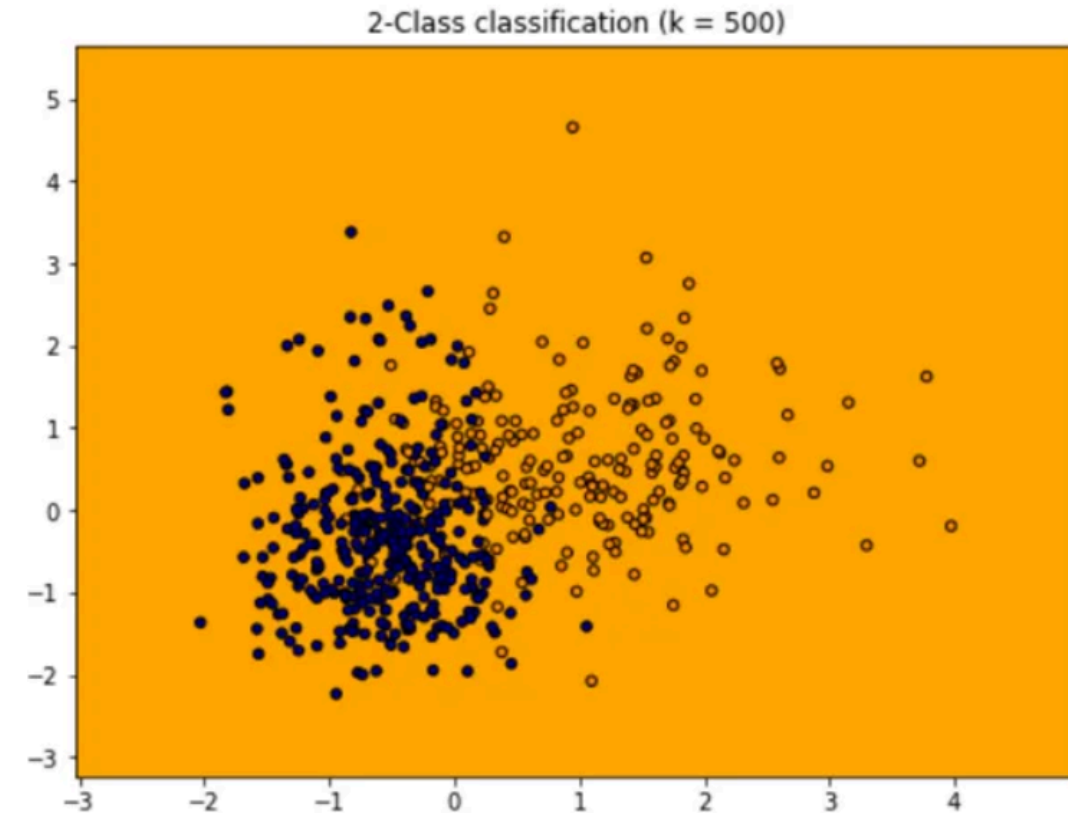


# Complejidad de un Modelo

## Overfitting



## Underfitting

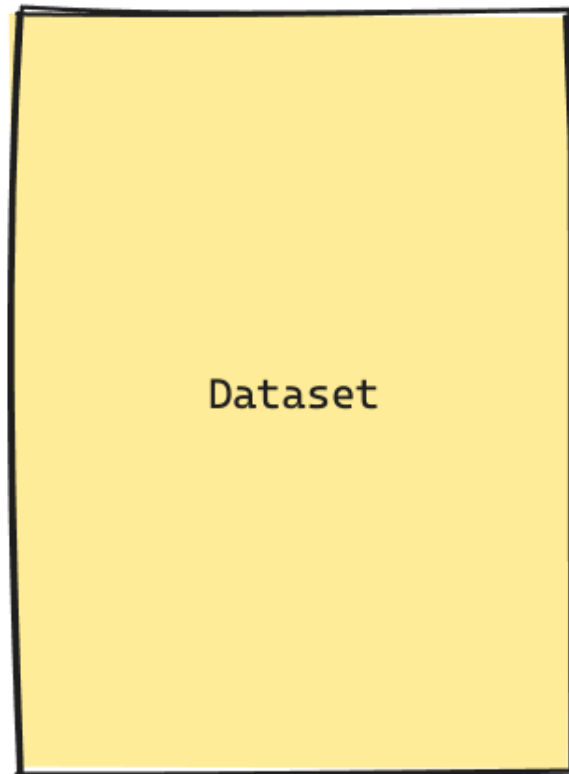


**Regularización:** Se refiere a una penalización que damos al modelo para que **disminuya su complejidad** de modo tal que evite el **Overfitting**. En el caso de KNN,  $K$  es un parámetro de regularización.

# Esquemas de Validación

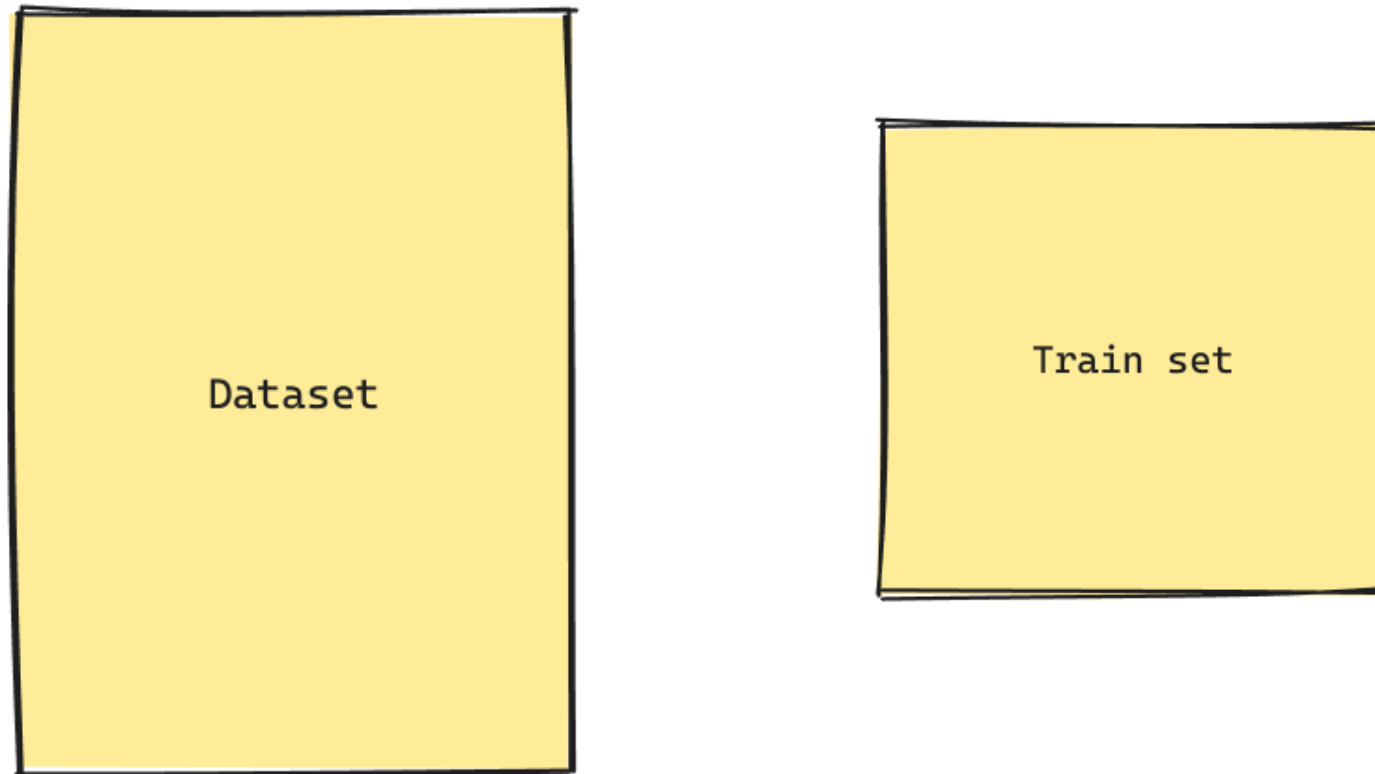
# Validación Cruzada: Holdout

También es conocido como **Train Test Split** o simplemente **Split**. Corresponde a la separación de nuestra data cuando con el propósito de aislar observaciones que el modelo no vea para una correcta evaluación.



# Validación Cruzada: Holdout

También es conocido como **Train Test Split** o simplemente **Split**. Corresponde a la separación de nuestra data cuando con el propósito de aislar observaciones que el modelo no vea para una correcta evaluación.

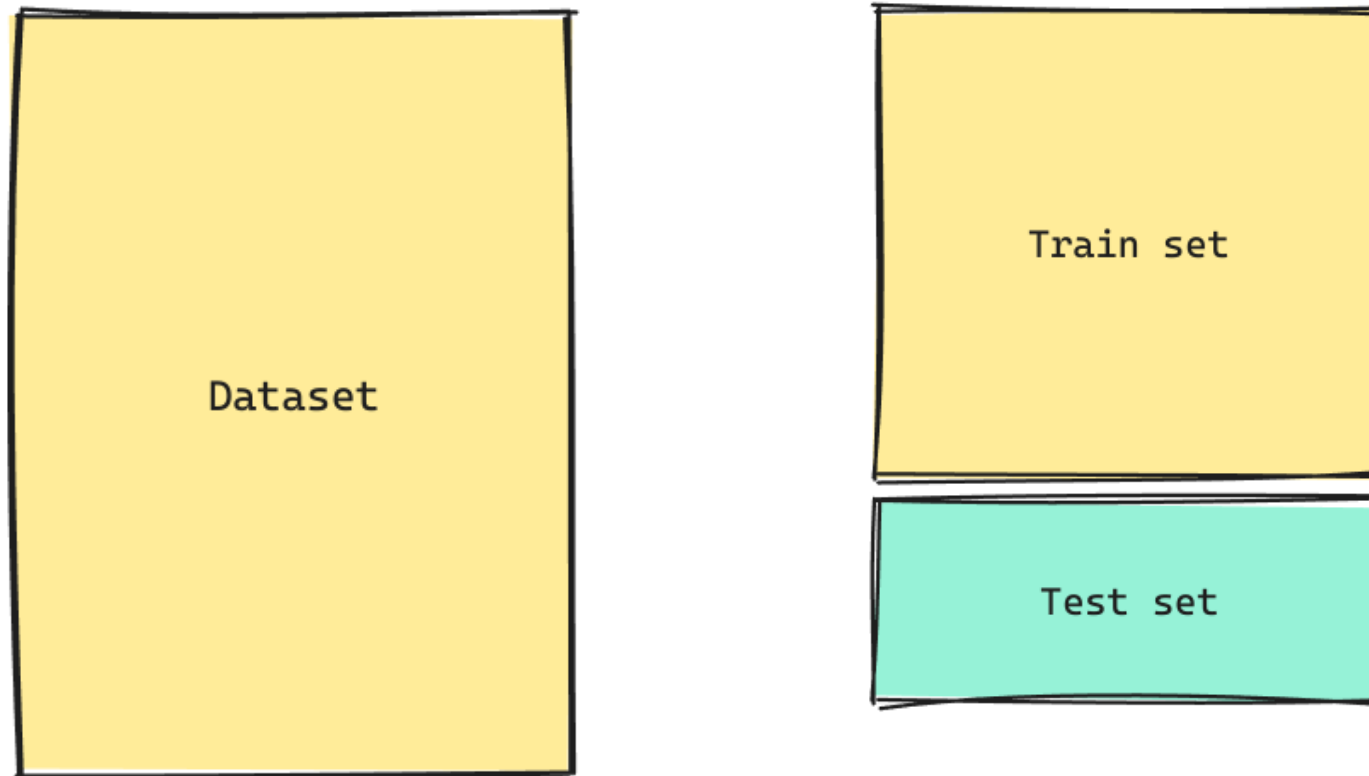


- El **train set** es la porción de los datos que se utilizará **exclusivamente** para entrenar los datos.



# Validación Cruzada: Holdout

También es conocido como **Train Test Split** o simplemente **Split**. Corresponde a la separación de nuestra data cuando con el propósito de aislar observaciones que el modelo no vea para una correcta evaluación.



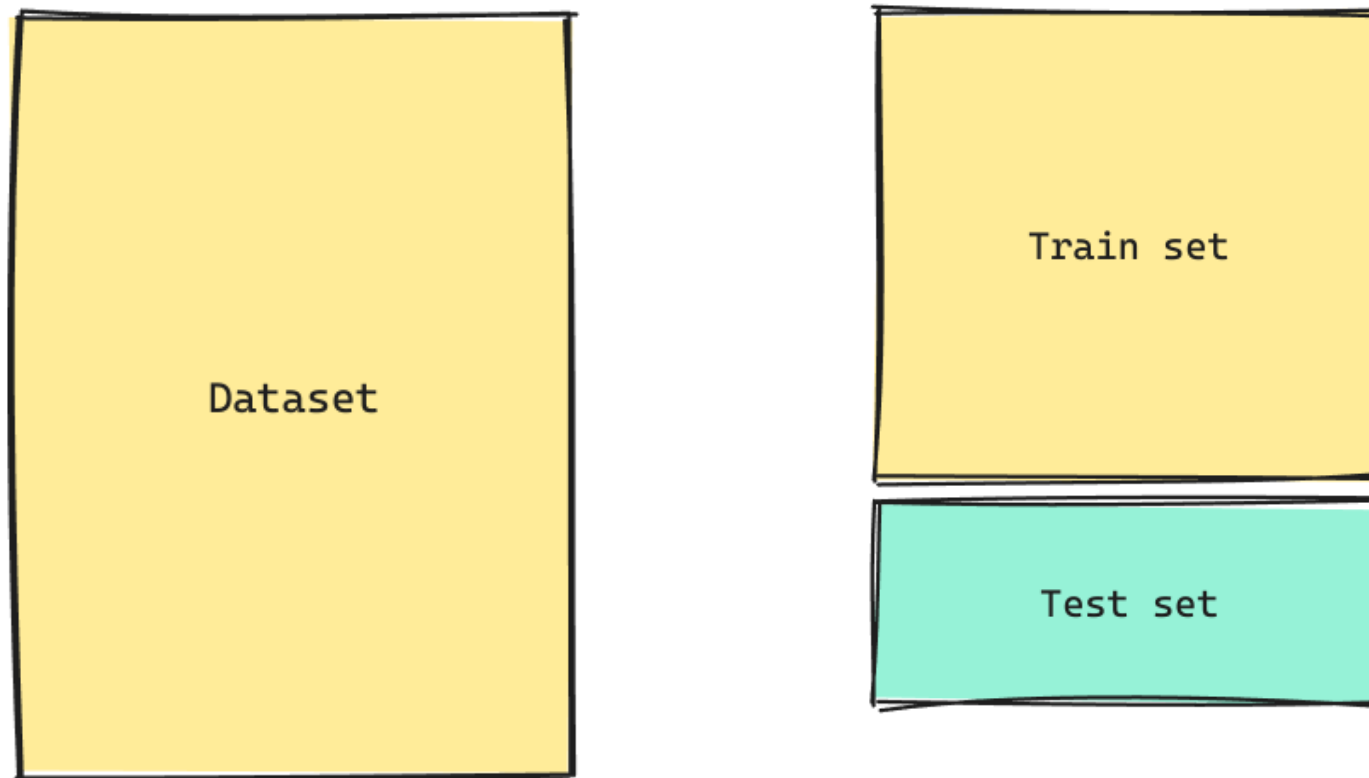
- El **train set** es la porción de los datos que se utilizará **exclusivamente** para entrenar los datos.



- El **test set** es la porción de los datos que se utilizará exclusivamente para validar los datos.
- El **test set** simula los datos que eventualmente entrarán el modelo para obtener una predicción.

# Validación Cruzada: Holdout

También es conocido como **Train Test Split** o simplemente **Split**. Corresponde a la separación de nuestra data cuando con el propósito de aislar observaciones que el modelo no vea para una correcta evaluación.



- El **train set** es la porción de los datos que se utilizará **exclusivamente** para entrenar los datos.



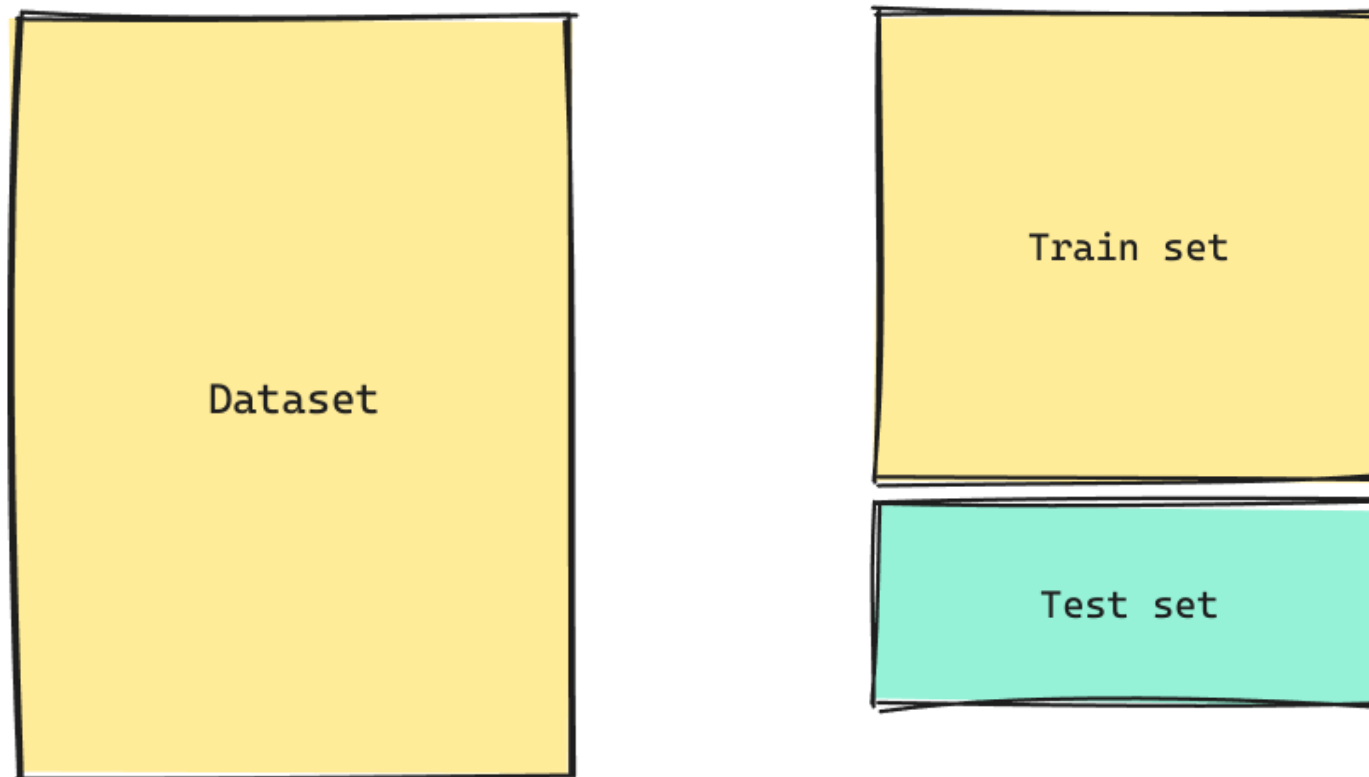
- El **test set** es la porción de los datos que se utilizará exclusivamente para validar los datos.
- El **test set** simula los datos que eventualmente entrarán el modelo para obtener una predicción.



- Normalmente se utilizan splits del tipo 70/30, 80/20 o 90/10.

# Validación Cruzada: Holdout

También es conocido como **Train Test Split** o simplemente **Split**. Corresponde a la separación de nuestra data cuando con el propósito de aislar observaciones que el modelo no vea para una correcta evaluación.



- El **train set** es la porción de los datos que se utilizará **exclusivamente** para entrenar los datos.



- El **test set** es la porción de los datos que se utilizará exclusivamente para validar los datos.
- El **test set** simula los datos que eventualmente entrarán el modelo para obtener una predicción.



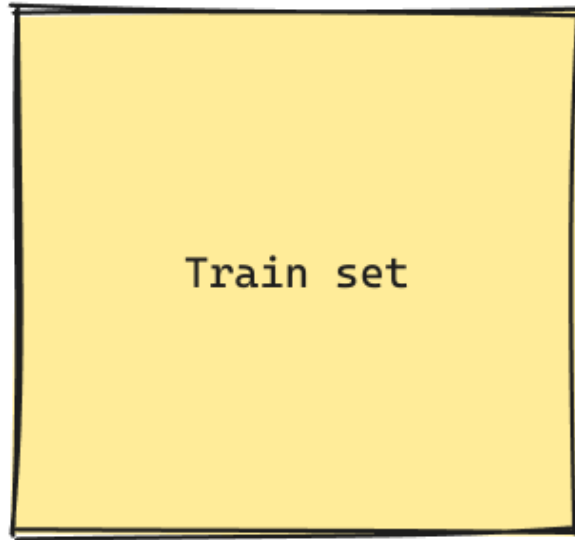
- Normalmente se utilizan splits del tipo 70/30, 80/20 o 90/10.



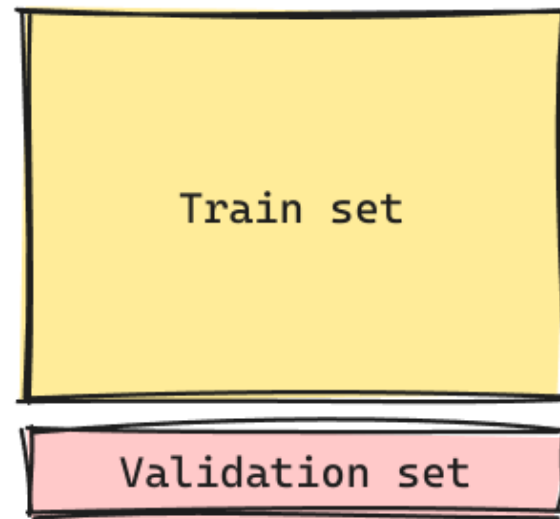
- ¿Cuál es el problema con este tipo de validación?

# Variante Holdout

# Variante Holdout

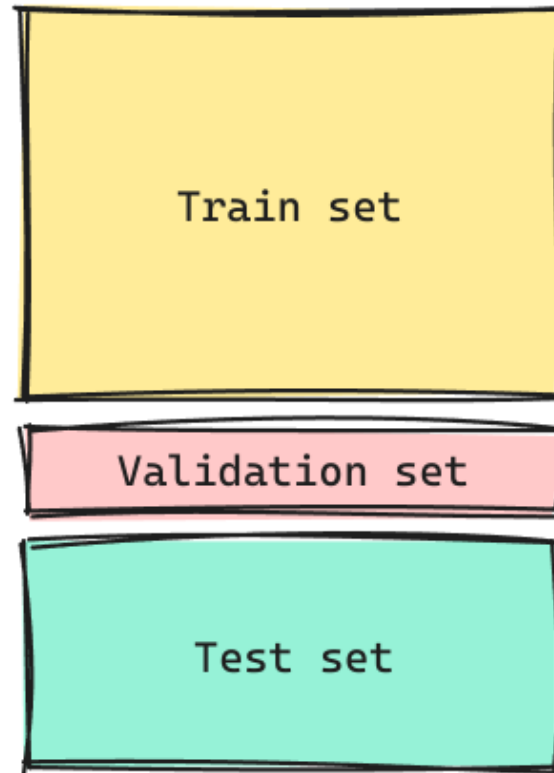


# Variante Holdout



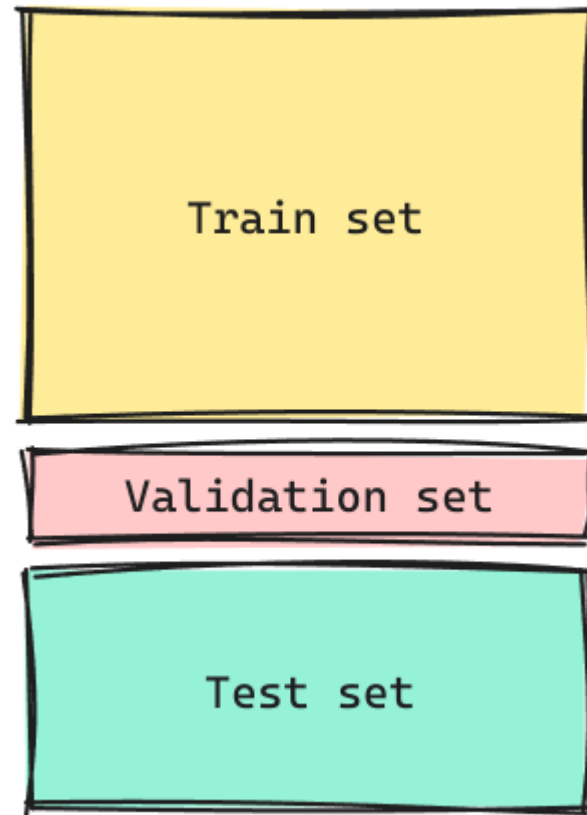
- ⚠ • Se agrega un **validation set** el cuál se utilizará para escoger los **hiperparámetros** que muestren un mejor poder de generalización.

# Variante Holdout



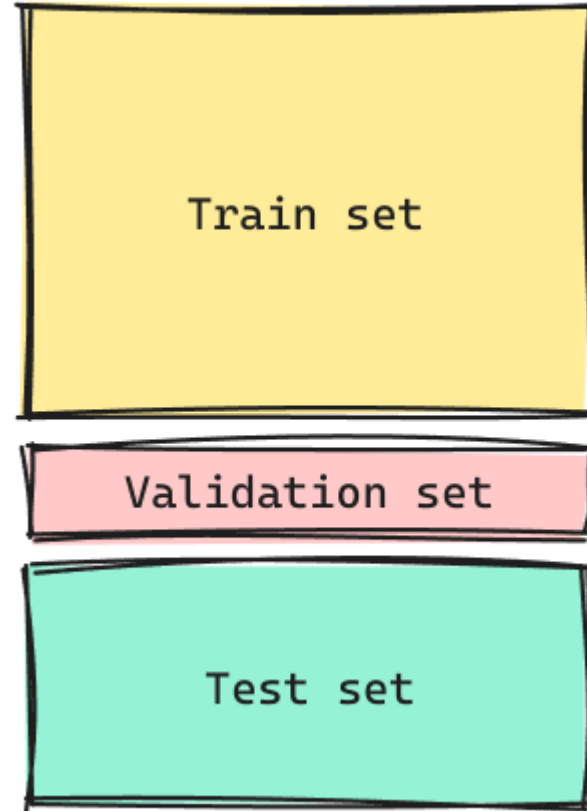
- ⚠ • Se agrega un **validation set** el cuál se utilizará para escoger los **hiperparámetros** que muestren un mejor poder de generalización.
- 💡 • El **train set** y el **test set** cumplen la misma función que tenían antes.

# Variante Holdout: Procedimiento





# Variante Holdout: Procedimiento



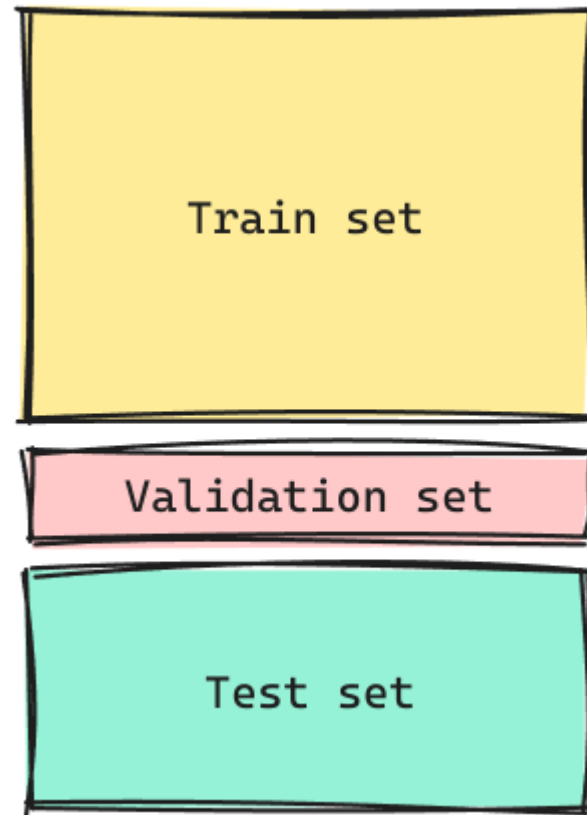
## Procedimiento

- Repetir para cada **Modelo** a probar.



- Vamos a entender un modelo como la combinación de un **Algoritmo de Aprendizaje** + **Hiperparámetros** + **Preprocesamiento**.

# Variante Holdout: Procedimiento



## Procedimiento

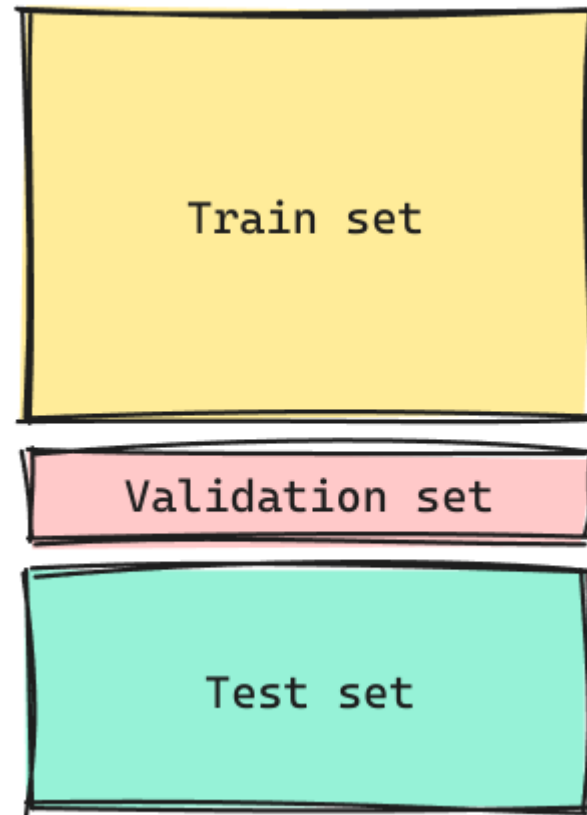
- Repetir para cada **Modelo** a probar.



- Vamos a entender un modelo como la combinación de un **Algoritmo de Aprendizaje** + **Hiperparámetros** + **Preprocesamiento**.

1. Se entrena cada **Modelo** en el **train set**. Se mide una métrica de Evaluación apropiada utilizando el **Validation Set**. La llamaremos *métrica de Validación*.

# Variante Holdout: Procedimiento



## Procedimiento

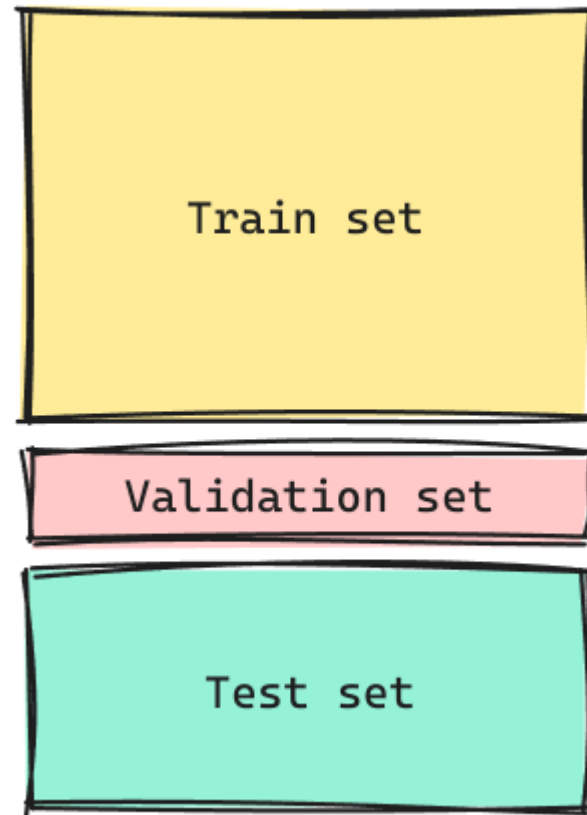
- Repetir para cada **Modelo** a probar.



- Vamos a entender un modelo como la combinación de un **Algoritmo de Aprendizaje** + **Hiperparámetros** + **Preprocesamiento**.

1. Se entrena cada **Modelo** en el **train set**. Se mide una métrica de Evaluación apropiada utilizando el **Validation Set**. La llamaremos *métrica de Validación*.
2. Se escoge el mejor **Modelo** como el que tenga la mejor *métrica de Validación*.

# Variante Holdout: Procedimiento



## Procedimiento

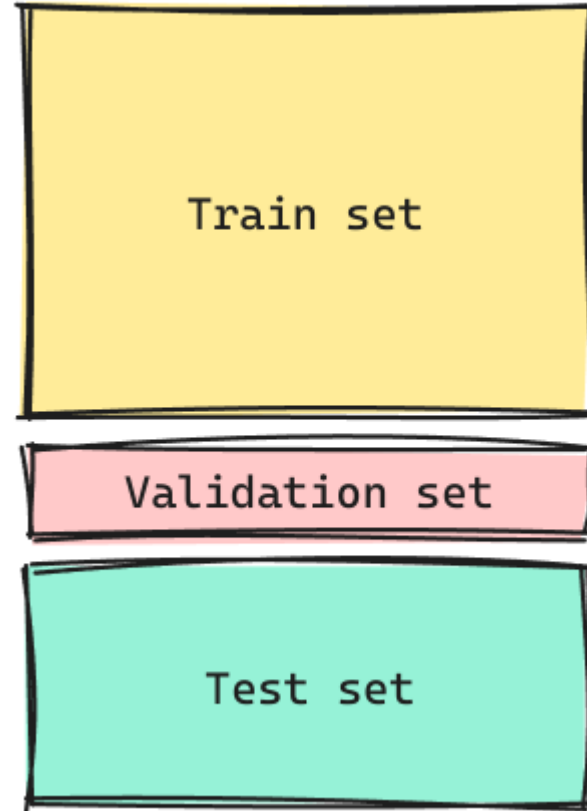
- Repetir para cada **Modelo** a probar.



• Vamos a entender un modelo como la combinación de un **Algoritmo de Aprendizaje** + **Hiperparámetros** + **Preprocesamiento**.

1. Se entrena cada **Modelo** en el **train set**. Se mide una métrica de Evaluación apropiada utilizando el **Validation Set**. La llamaremos *métrica de Validación*.
2. Se escoge el mejor **Modelo** como el que tenga la mejor *métrica de Validación*.
3. Se reentrena el *modelo escogido* pero ahora en un “nuevo set” compuesto por el **Train set** + el **Validation set**.

# Variante Holdout: Procedimiento



## Procedimiento

- Repetir para cada **Modelo** a probar.




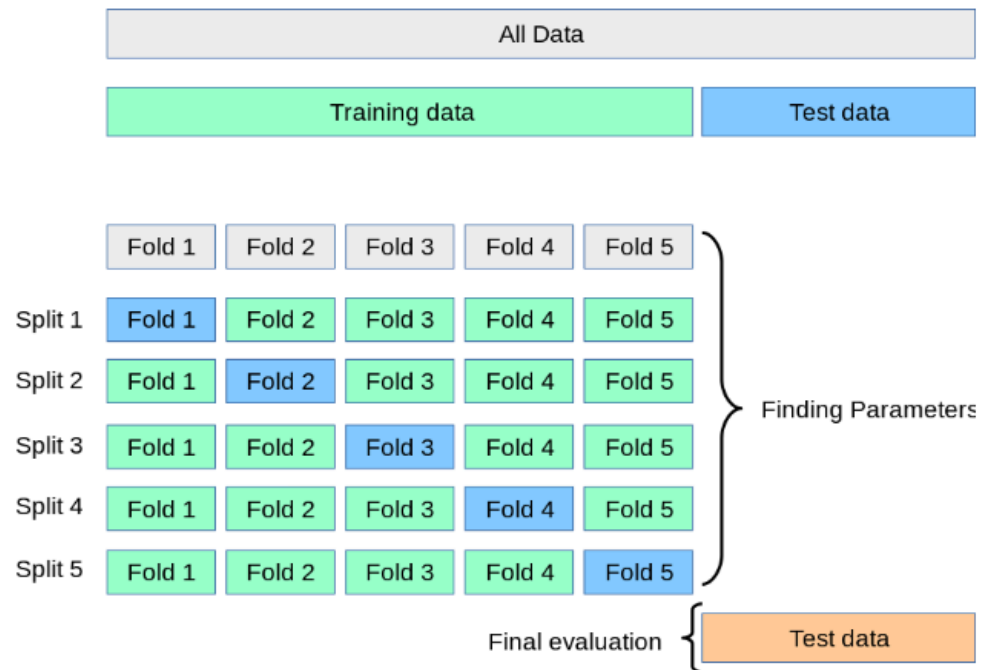
• Vamos a entender un modelo como la combinación de un **Algoritmo de Aprendizaje** + **Hiperparámetros** + **Preprocesamiento**.

1. Se entrena cada **Modelo** en el **train set**. Se mide una métrica de Evaluación apropiada utilizando el **Validation Set**. La llamaremos *métrica de Validación*.
2. Se escoge el mejor **Modelo** como el que tenga la mejor *métrica de Validación*.
3. Se reentrena el *modelo escogido* pero ahora en un “nuevo set” compuesto por el **Train set** + el **Validation set**.
4. Se reporta el rendimiento final del *mejor modelo* (al momento del diseño) utilizando métricas medidas en el **Test Set**.

# K-Fold CV

- El proceso de Holdout podría llevar a un proceso de overfitting del Test Set si el modelo no es lo suficientemente robusto.
- Definiremos la robustez más adelante.

 El K-Fold CV se aplica sólo al **Train Set** y la métrica final que se reporta utilizando el **Test Set**.



## Fold

Entenderemos Folds como divisiones que haremos a nuestro dataset. (En el ejemplo se divide el dataset en 5 Folds).

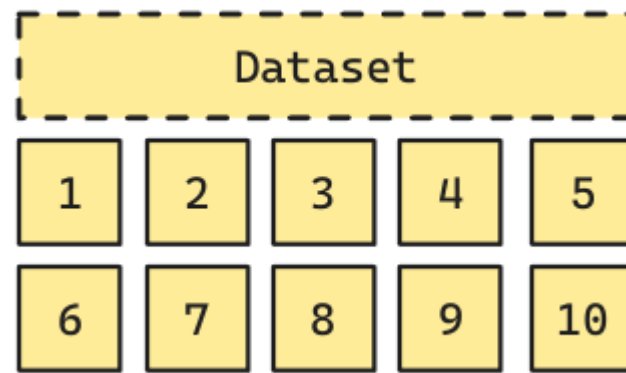
## Split

Entenderemos Splits, como iteraciones. En cada iteración utilizaremos un Fold como **Validation Set** y todos los Folds restantes como **Train Set**.

- La métrica final se calculará como el promedio de las **Métricas de Validación** para cada **Split**.
- A veces la variabilidad (medido a través de la Desviación Estándar) también es usado como criterio para elegir el mejor modelo.

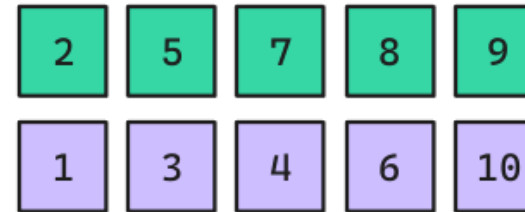
# Bootstrap

Consiste en generar subgrupos aleatorios con repetición. Normalmente requiere especificar el tamaño de la muestra de entrenamiento. Y la cantidad de repeticiones que del proceso. Los sets de validación (en morado) acá se denominan *out-of-bag* samples.



- La métrica final a reportar se mide como el promedio de los *out-of-bag* samples.

Iter 1:



Iter 2:



Iter 3:



# Variantes y Consejos

## Stratified K-Fold

Es la variante más utilizada de K-Fold el cual genera los folds considerando que se mantenga la proporción de etiquetas en cada Fold.

## Leave One Out

Sería una variante con  $K = n$ . Por lo tanto, el **Validation Set** tiene sólo una observación.


### 💡 ¿Cuándo usar cada uno?

- Si se tiene una cantidad de datos suficiente (normalmente tamaños muy grandes se prefiere) el **Holdout**.
  - Entre más registros, menos % de **Validation Set** se deja.
- Si se requiere robustez, o hay Test sets que son muy variables se prefiere **K-Fold**.
  - Si es que hay desbalance de clases, se prefiere la versión **Stratified**.
- Si se tienen *muy pocos* datos, entonces utilizar **Leave-One-Out**.
- **Bootstrap** también es utilizado cuando se tengan pocos datos. Aunque suele ser un approach más estadístico.



# Baseline

Un modelo **Base line** es un modelo simple, normalmente sin aprendizaje asociado o con poder de aprendizaje más limitado, el cuál será utilizado como medida de referencia para ver si algoritmos más complejos efectivamente están aprendiendo.

 Si estamos probando un nuevo modelo y éste es capaz de **superar el rendimiento de un Baseline**, se considera como que estamos aprendiendo algo nuevo.

 Modelos que **no superaron** el puntaje de un modelo Baseline normalmente son deshechados.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train,y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train,y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train, y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train,y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train, y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Implementación en Python: Baselines

```
1 from sklearn.dummy import DummyClassifier
2
3 dc = DummyClassifier(strategy="prior", random_state = 42, constant=None)
4 dc.fit(X_train, y_train)
5 y_pred = dc.predict(X_test)
```

- **strategy:** Corresponde a estrategias “dummy” con las cuales generar predicciones.
  - “prior”: predice siempre la clase más frecuente observada en el entrenamiento. Si se predice la probabilidad, se devuelve la probabilidad empírica.
  - “constant”: Devuelve un valor constante provisto por el usuario.
  - “uniform”: Predice probabilidades aleatorias obtenidas mediante una distribución uniforme.

# Data Leakage

## Fuga de Datos

Se refiere al proceso donde el modelo por alguna razón conoce información que no debería conocer. Puede ser información del **Test Set** o variables que revelan información primordial sobre la etiqueta.

⚠ Cuando existe **Data Leakage** es posible que los resultados del modelo no reflejen correctamente su rendimiento dando una falsa sensación de optimismo.

## Ejemplos

- Estandarizar o aplicar preprocesamientos antes del Split de la Data.
- Utilizar variables que tienen directa relación con el Target.



# Data Leakage

## Fuga de Datos

Se refiere al proceso donde el modelo por alguna razón conoce información que no debería conocer. Puede ser información del **Test Set** o variables que revelan información primordial sobre la etiqueta.

⚠ Cuando existe **Data Leakage** es posible que los resultados del modelo no reflejen correctamente su rendimiento dando una falsa sensación de optimismo.

## Ejemplos

- Estandarizar o aplicar preprocesamientos antes del Split de la Data.
- Utilizar variables que tienen directa relación con el Target.

💡 Se recomienda siempre que sea posible utilizar **Pipelines** para poder evitar el **Data Leakage**.

# Arrivederci!