



**Mondragon
Unibertsitatea**

Goi Eskola
Politeknikoa

Fundamentos para la validación de modelos

Fundamentos del Aprendizaje
Automático

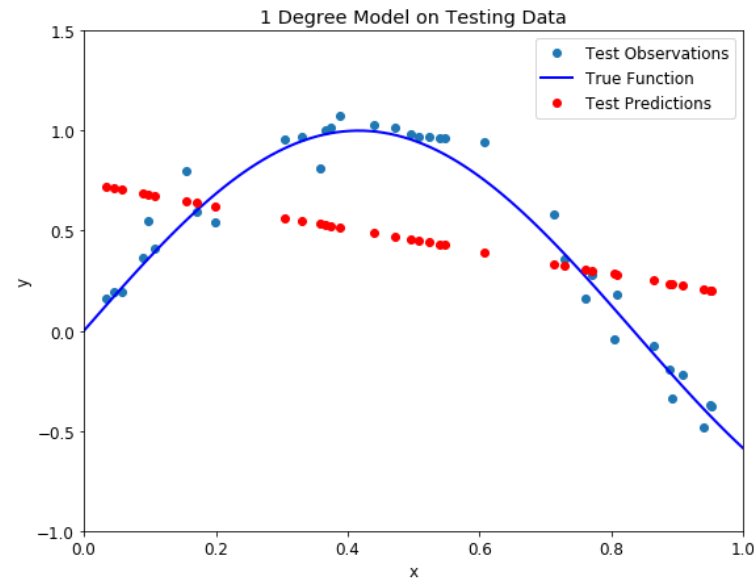
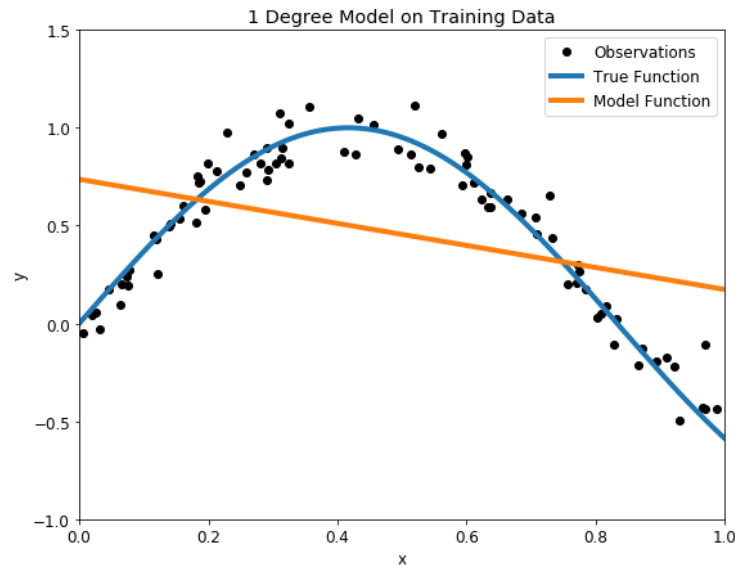
Validación de modelos

- 1) Estabilidad
- 2) ¿El modelo generaliza con nuevos datos?
 - Capturamos patrones importantes
 - Reducir el efecto del “ruido”

¿En qué nos ayuda?

1. Evaluar la calidad del modelo
2. Seleccionar el modelo que mejor funcione con datos nuevos
3. Evitar overfitting y underfitting
 - I. **Overfitting:**
 - I. Captura de ruido
 - II. Captura de patrones que no generalizan bien con nuevos datos.
 - El modelo funciona extremadamente bien con los datos de entrenamiento
 - El modelo no da buenos resultados en el conjunto de datos de test.
 - II. **Underfitting:** no se capturan suficientes patrones en los datos. El modelo funciona pobremente en los datos de entrenamiento y validación

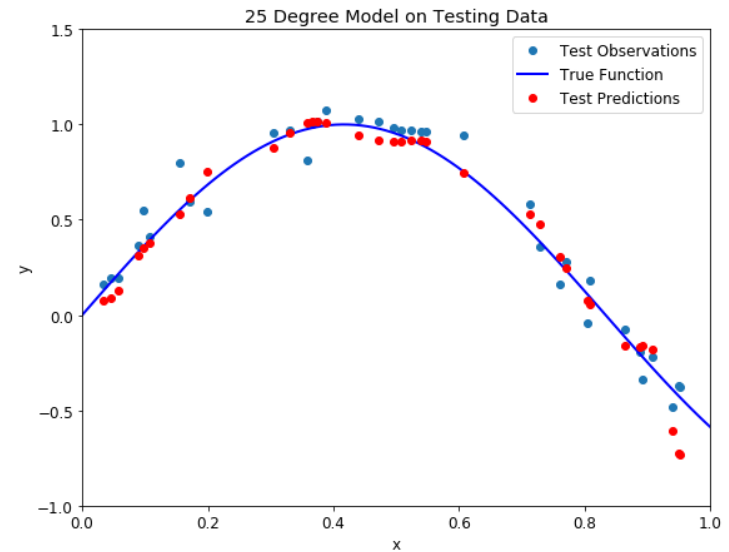
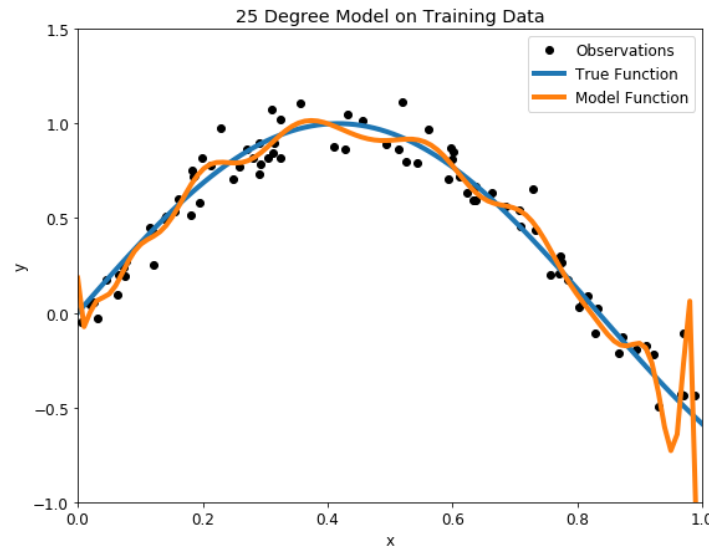
Underfitting



- Ejemplo: el modelo pasa directamente por los datos, sin tenerlos en cuenta!
 - Low **variance**:
 - El modelo no es dependiente de los datos
 - High **bias** (sesgo):
 - Se hace una fuerte asunción: los datos siguen una distribución lineal
 - Se sobresimplifica el modelo





Overfitting



- Ejemplo: el modelo tiene en cuenta todos los puntos de entrenamiento:
 - High **variance**:
 - El modelo es extremadamente dependiente de los datos => dependiente del ruido
 - El modelo cambiará significativamente si cambiamos los datos
 - Low **bias** (sesgo):
 - No se hace una fuerte asunción sobre los datos

Bias vs variance trade-off

•  Bias =>  Varianza

•  Varianza =>  Bias

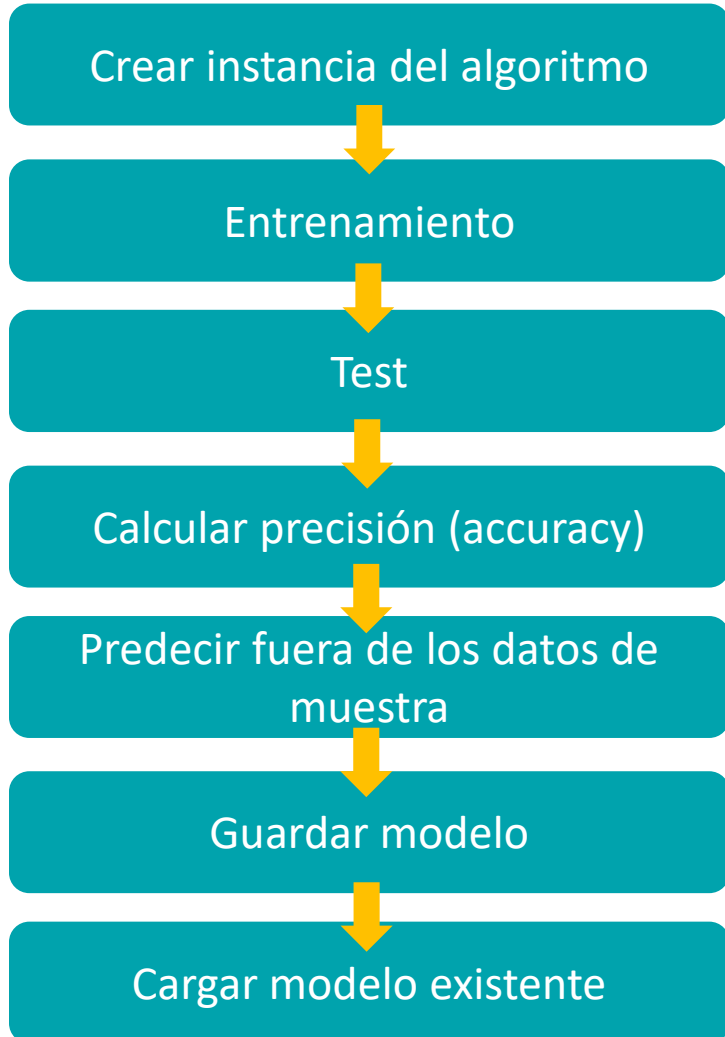
Objetivo



-Sesgo bajo
-Varianza baja

- Si tuviéramos que elegir...
 - ¿Elegiríamos el modelo con la mejor “puntuación” en el conjunto de entrenamiento?
- ¡Validación!

Proceso de creación de un modelo



Scikit -learn

```
knn = KNeighborsClassifier(n_neighbors=3)
```

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

```
y_pred = knn.predict(X_test)
```

```
print(metrics.accuracy_score(y_test, y_pred))
```

```
sample = [[3, 5, 4, 2], [2, 3, 5, 4]]  
preds = knn.predict(sample)
```

```
joblib.dump(knn, 'iris_knn.pkl')
```

```
knn = joblib.load('iris_knn.pkl')
```

Definiciones

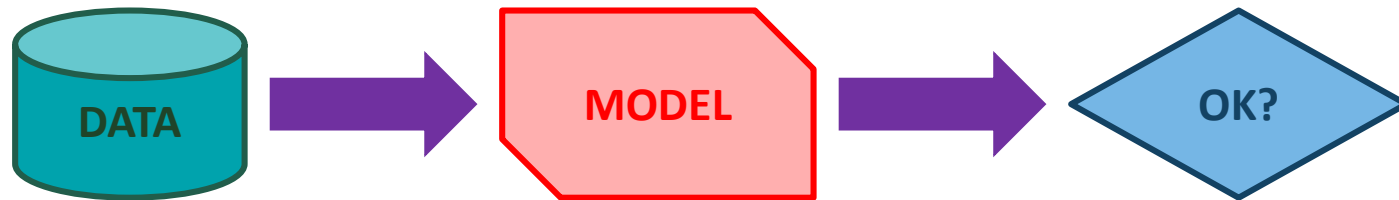
- Accuracy:
 - La proporción de clasificaciones correctas sobre todas las clasificaciones
 - Métrica fácil e intuitiva...
 - Pero problemática (to be seen...)

| | Predicted True | Predicted False |
|------------|----------------|-----------------|
| Real True | 850 | 35 |
| Real False | 45 | 70 |

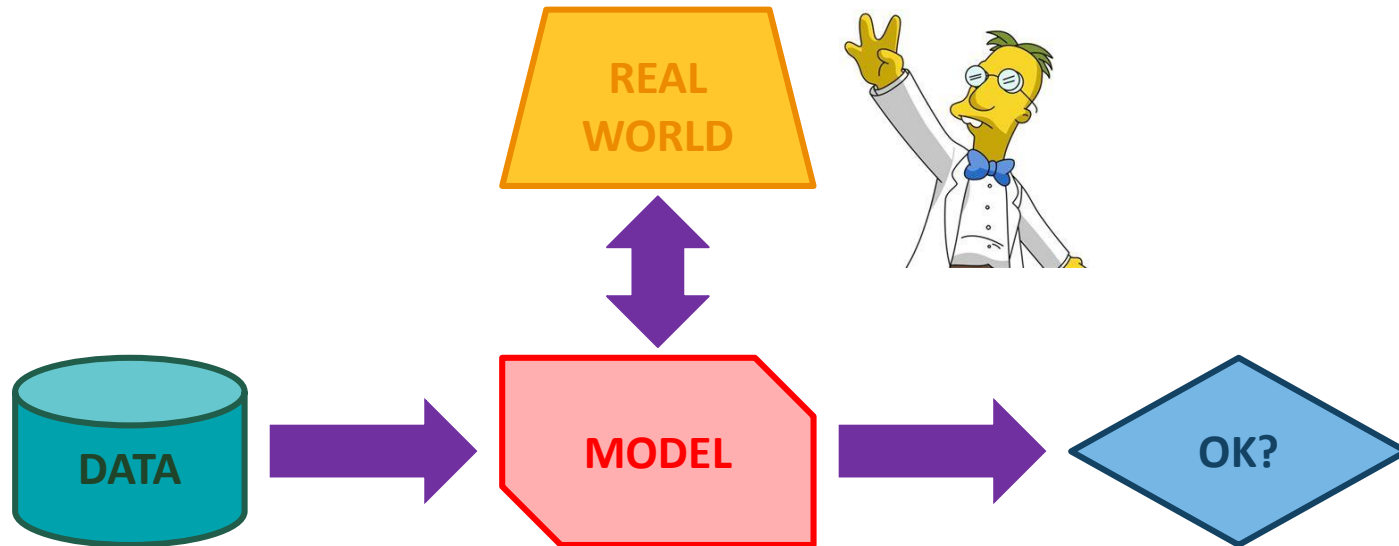
$$\frac{70 + 850}{1000} = 92\%$$

Definiciones

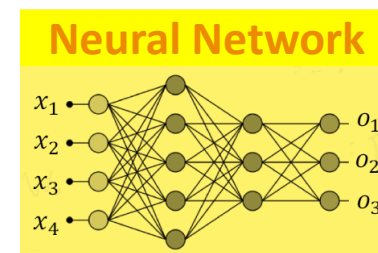
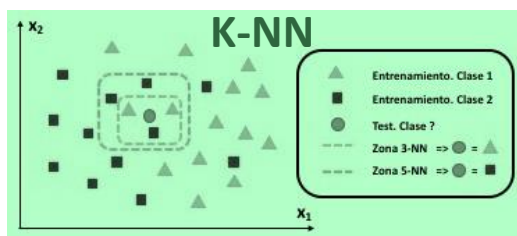
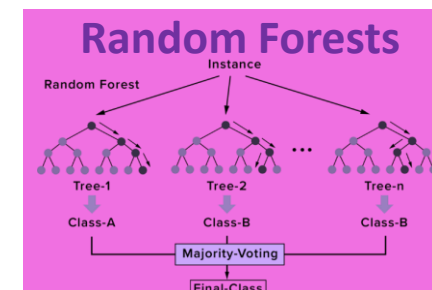
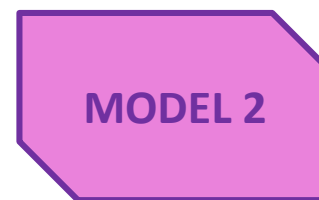
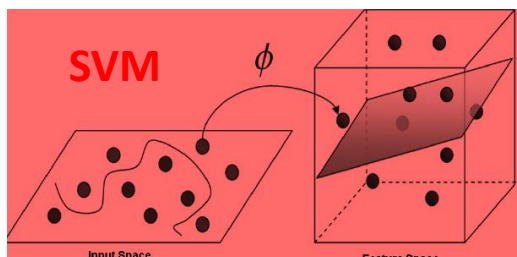
- El accuracy score no es buen indicador de la precisión cuando las clases no están balanceadas
 - Se debe completar con otros métodos y métricas de evaluación:
 - Matriz de confusión
 - F1 score (F1)
 - Especificidad
 - Área bajo la curva (Area under curve (ROC))



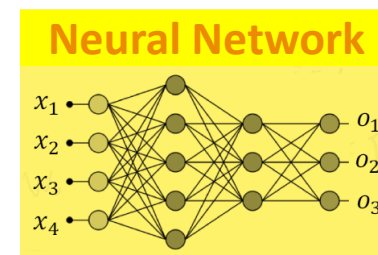
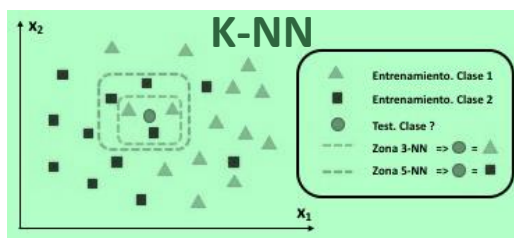
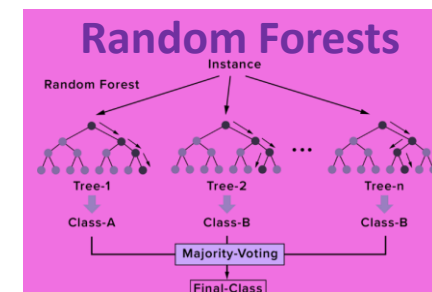
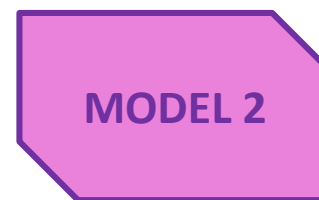
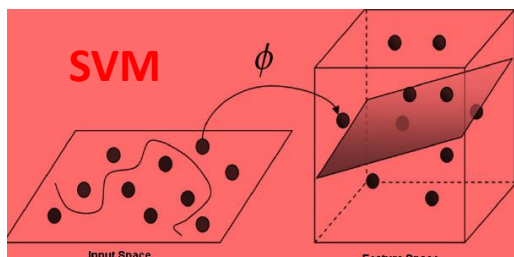
- ¿Cómo decidimos si un modelo es una **buena solución** a nuestro problema?



- Opciones:
 - Conocimiento experto
 - Automatización del trabajo de evaluación del experto



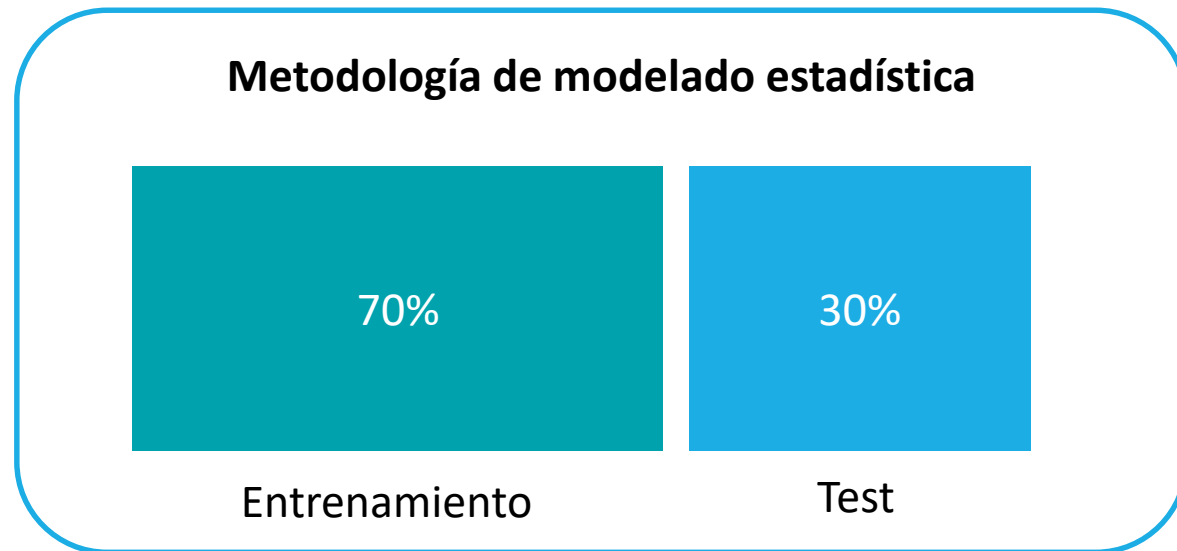
- Muchos **paradigmas diferentes** entre los que elegir
- Cada uno con uno o más **parámetros** que prefijar



- Muchos **paradigmas diferentes** entre los que elegir
- Cada uno con uno o más **parámetros** que prefijar

Conjuntos de entrenamiento y testeo

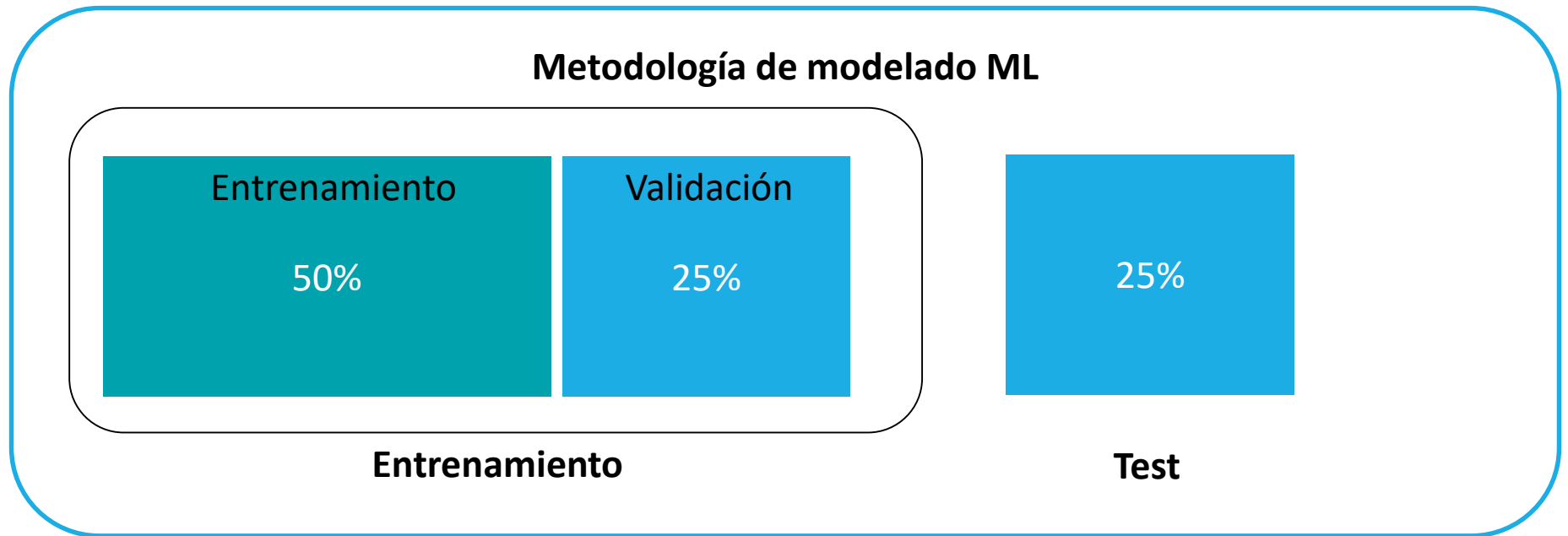
- Procedimiento estadístico:
 - Normalmente el conjunto de datos se divide aleatoriamente en conjunto de entrenamiento y test con un ratio de 70 - 30 o 80 - 10.



- Estadística: modelos robustos que son lineales por naturaleza: no hay problemas de varianza / sesgo altos.

Conjuntos de entrenamiento y testeo

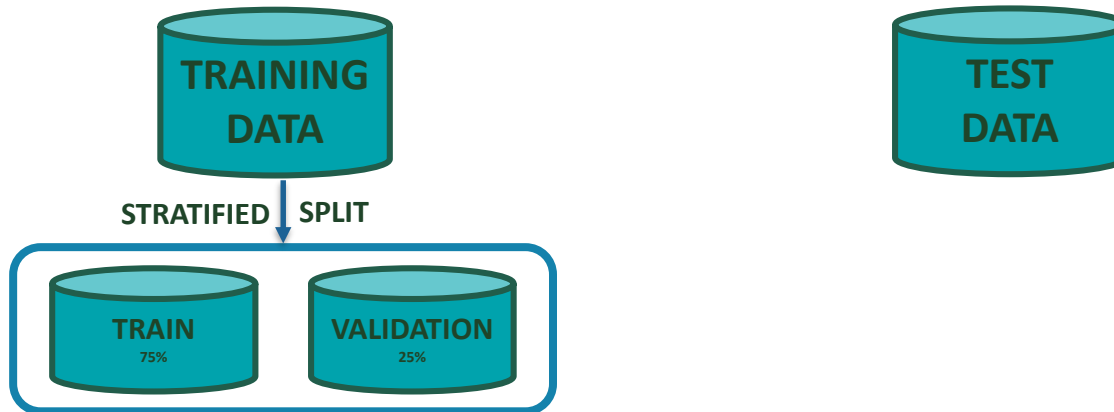
- ML: (Entrenamiento + validación) + test



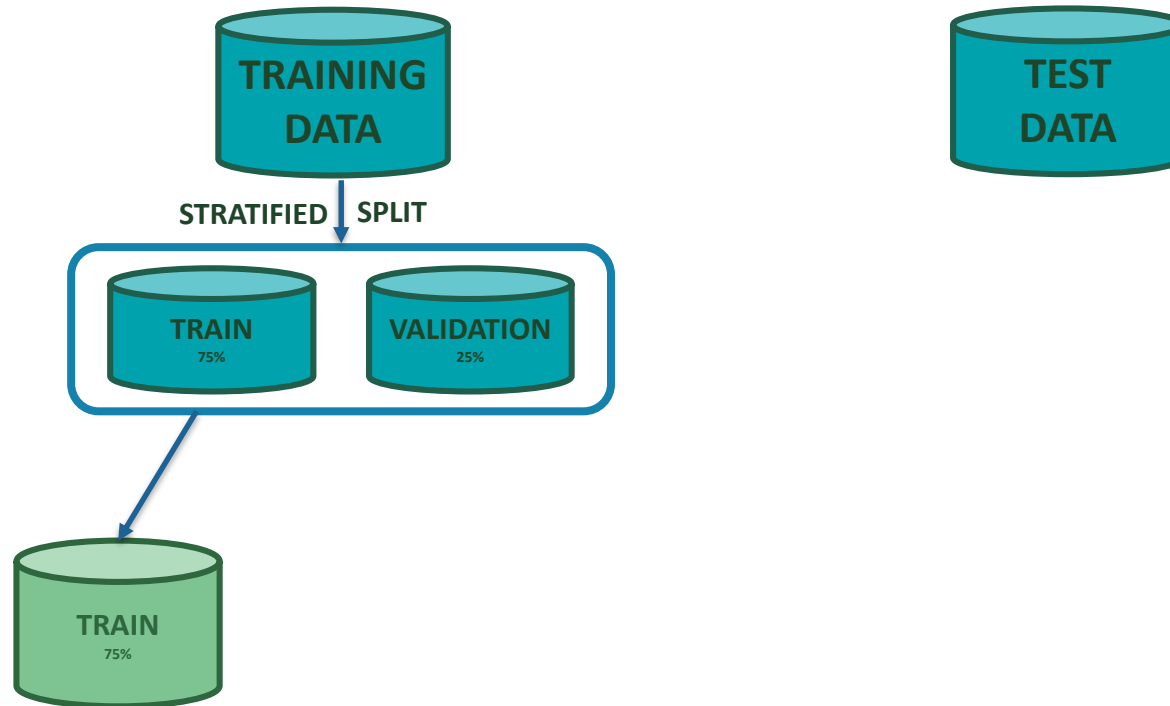
Conjuntos de entrenamiento y testeo



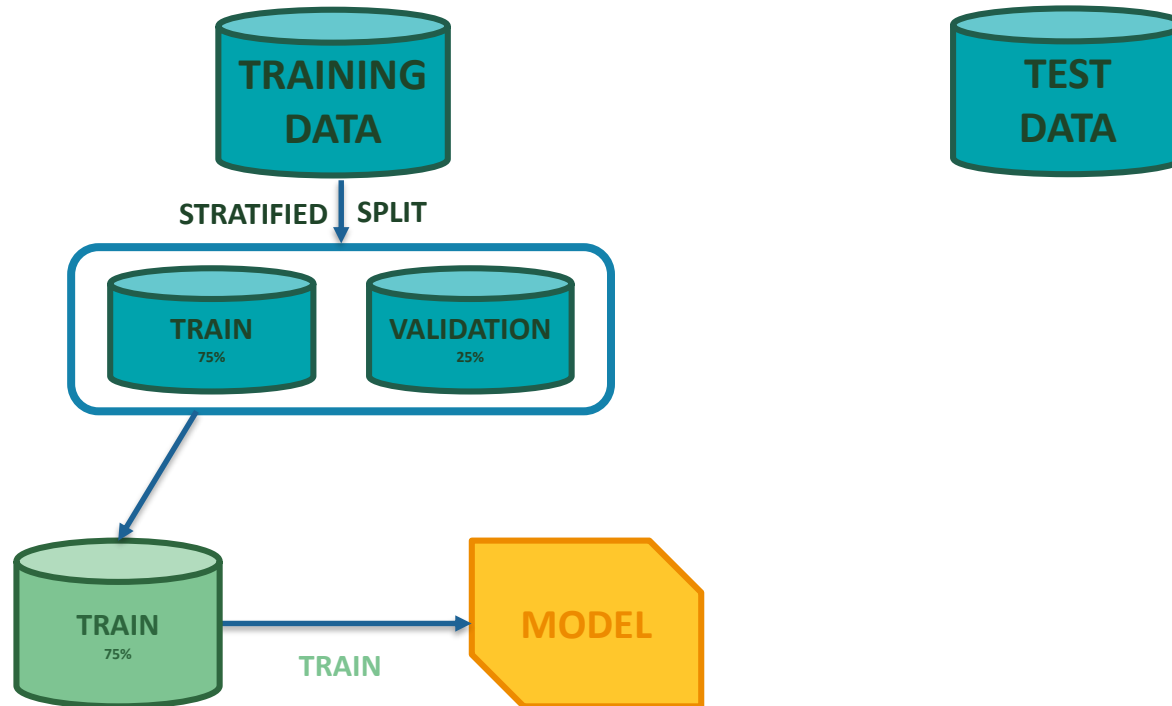
Conjuntos de entrenamiento y testeo



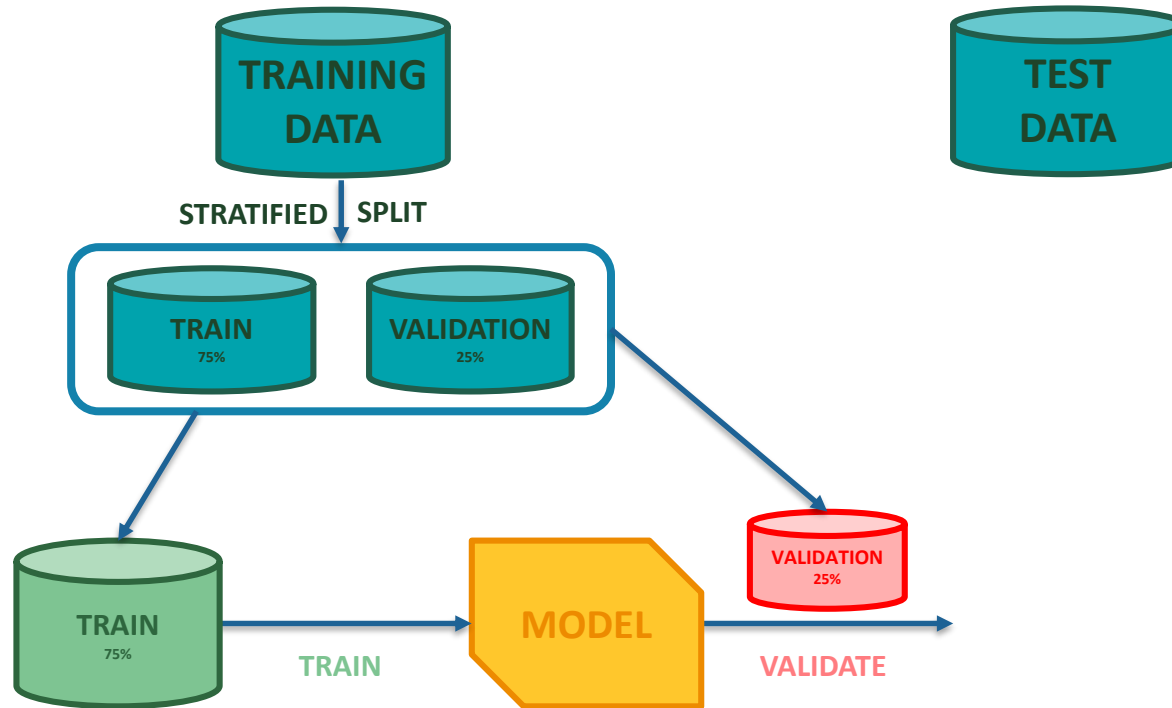
Conjuntos de entrenamiento y testeo



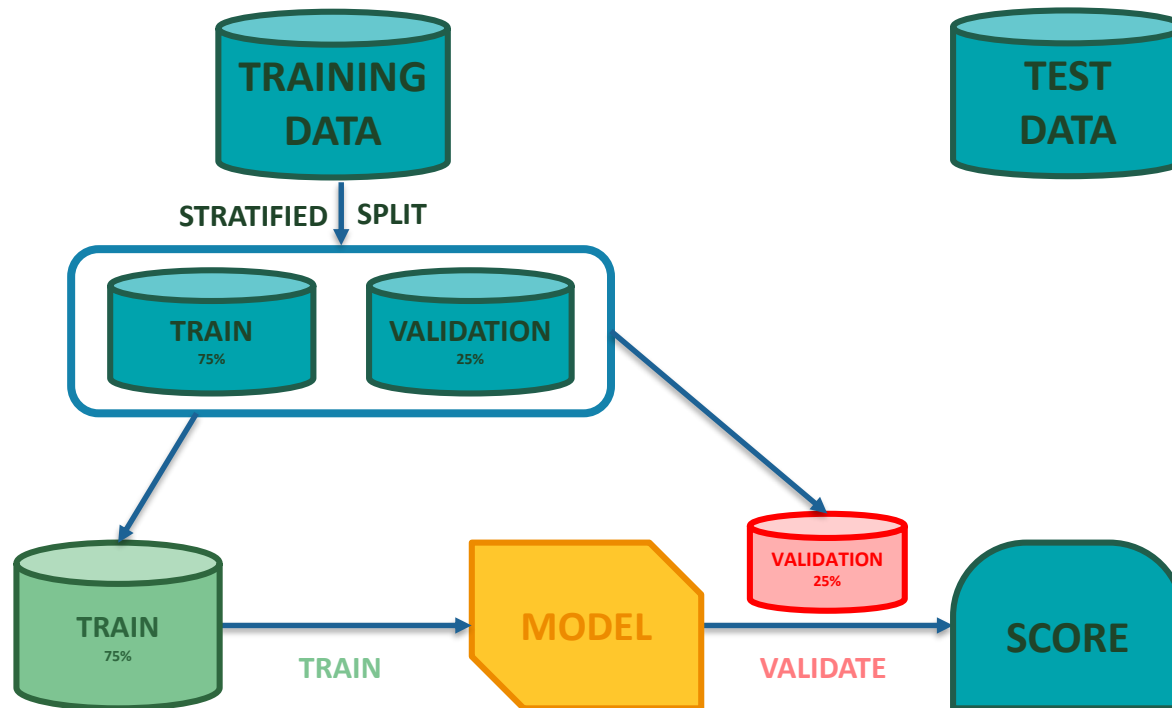
Conjuntos de entrenamiento y testeo



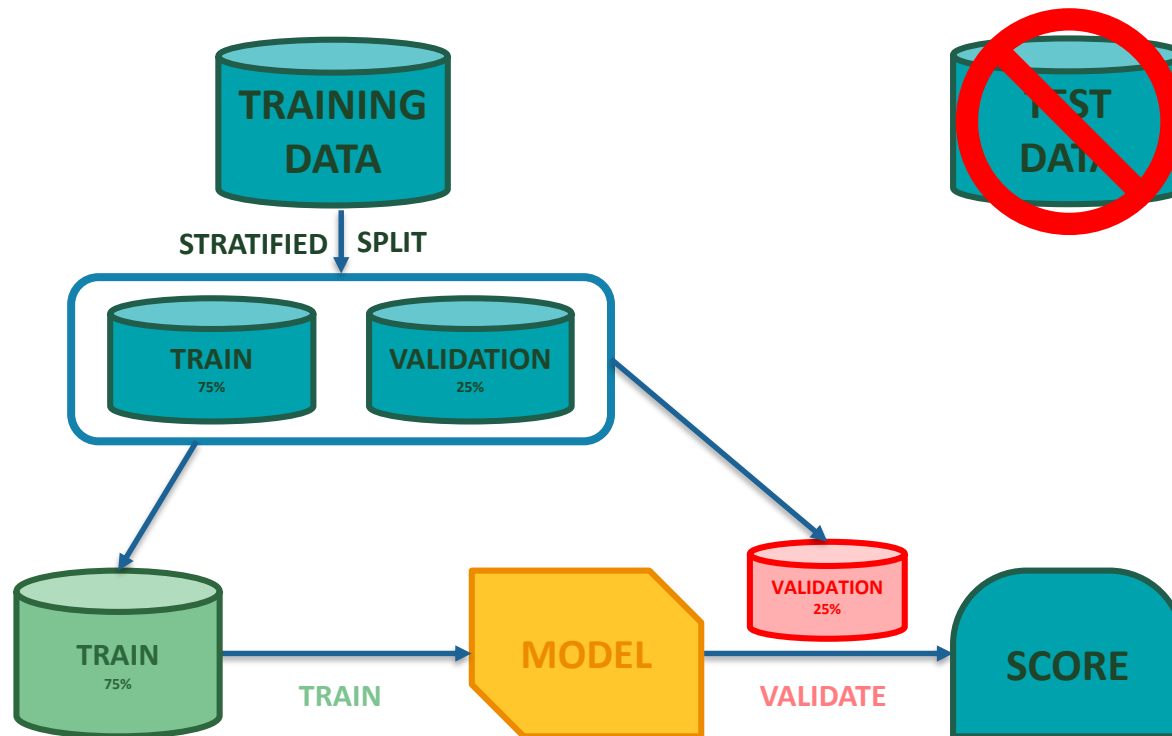
Conjuntos de entrenamiento y testeo



Conjuntos de entrenamiento y testeo



Conjuntos de entrenamiento y testeo



Conjuntos de entrenamiento y testeo

crear conjuntos de entrenamiento y test

```
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
(353, 10) (353,)
(89, 10) (89,)
```

entrenar el modelo

```
lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

Score => predictions vs y_test

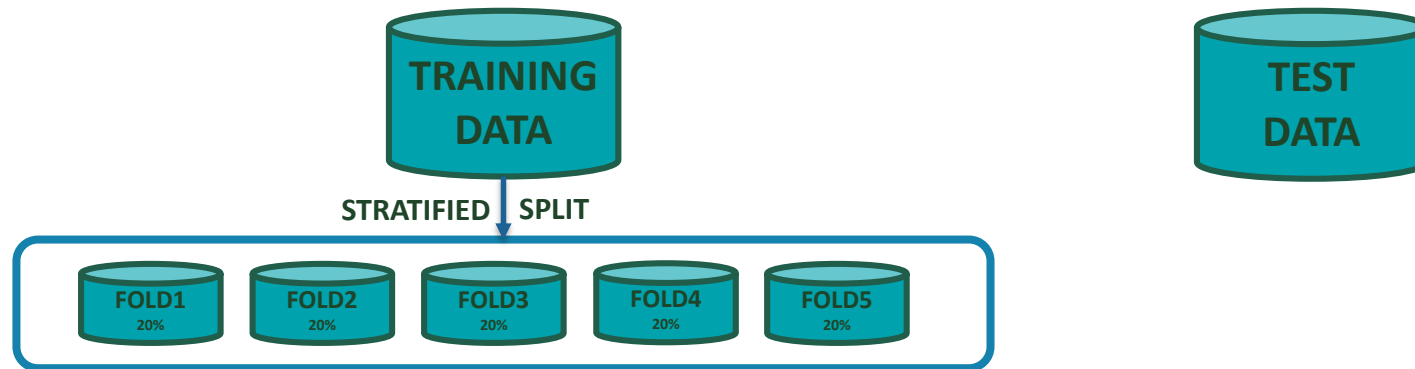
Cross-validation

- Objetivo:
 - Asegurar robustez
 - Normalmente:
 - Modelo creado en conjunto de entrenamiento
 - Evaluado en conjunto de test

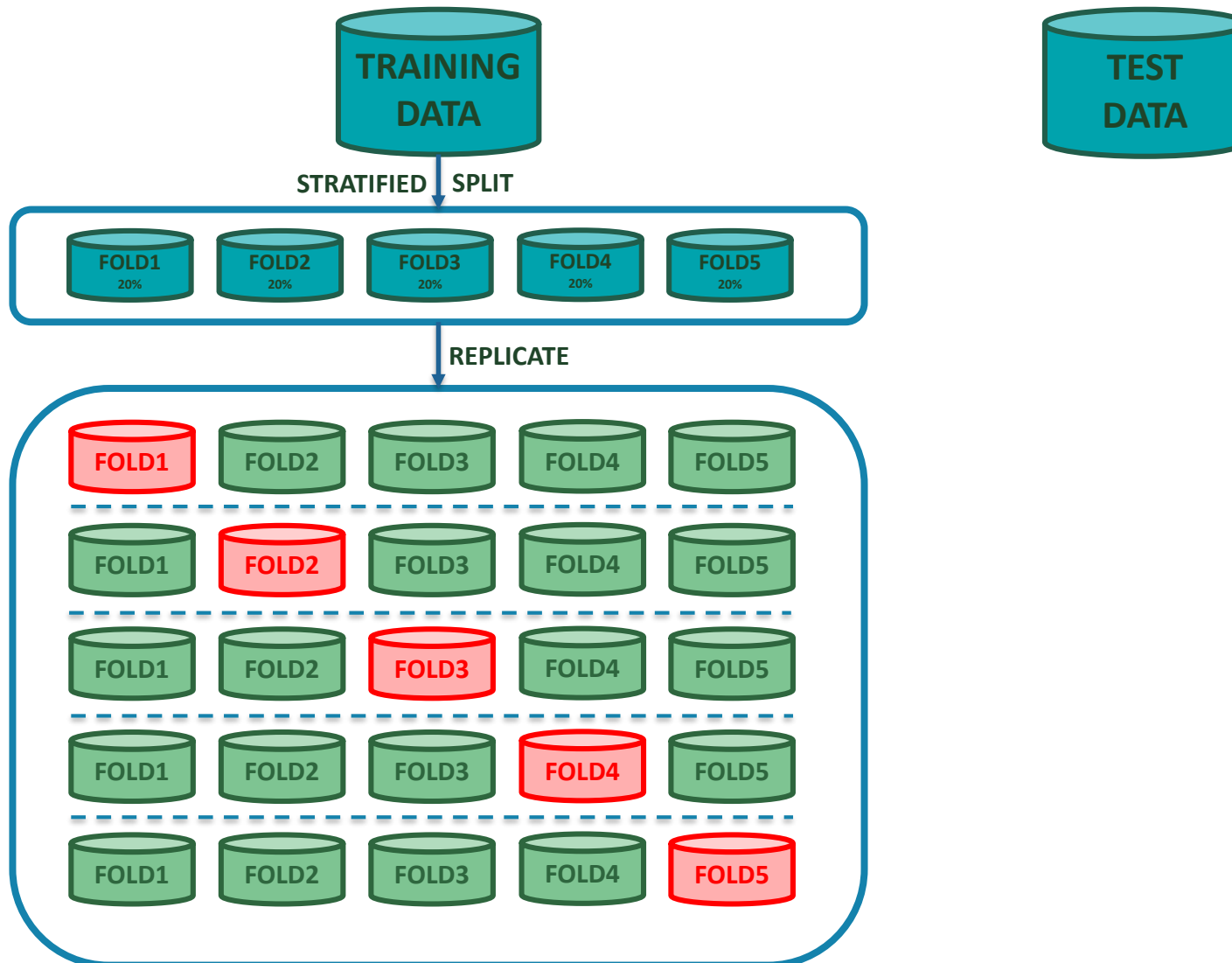
Cross-validation

- Objetivo:
 - Asegurar robustez
 - Normalmente:
 - Modelo creado en conjunto de entrenamiento
 - Evaluado en conjunto de test
 - Evitar sesgo en la muestra:
 - Datos fuera del entrenamiento
 - Complejidad del dataset
- Puede que los datos de entrenamiento y test no hayan sido seleccionados **homogéneamente**

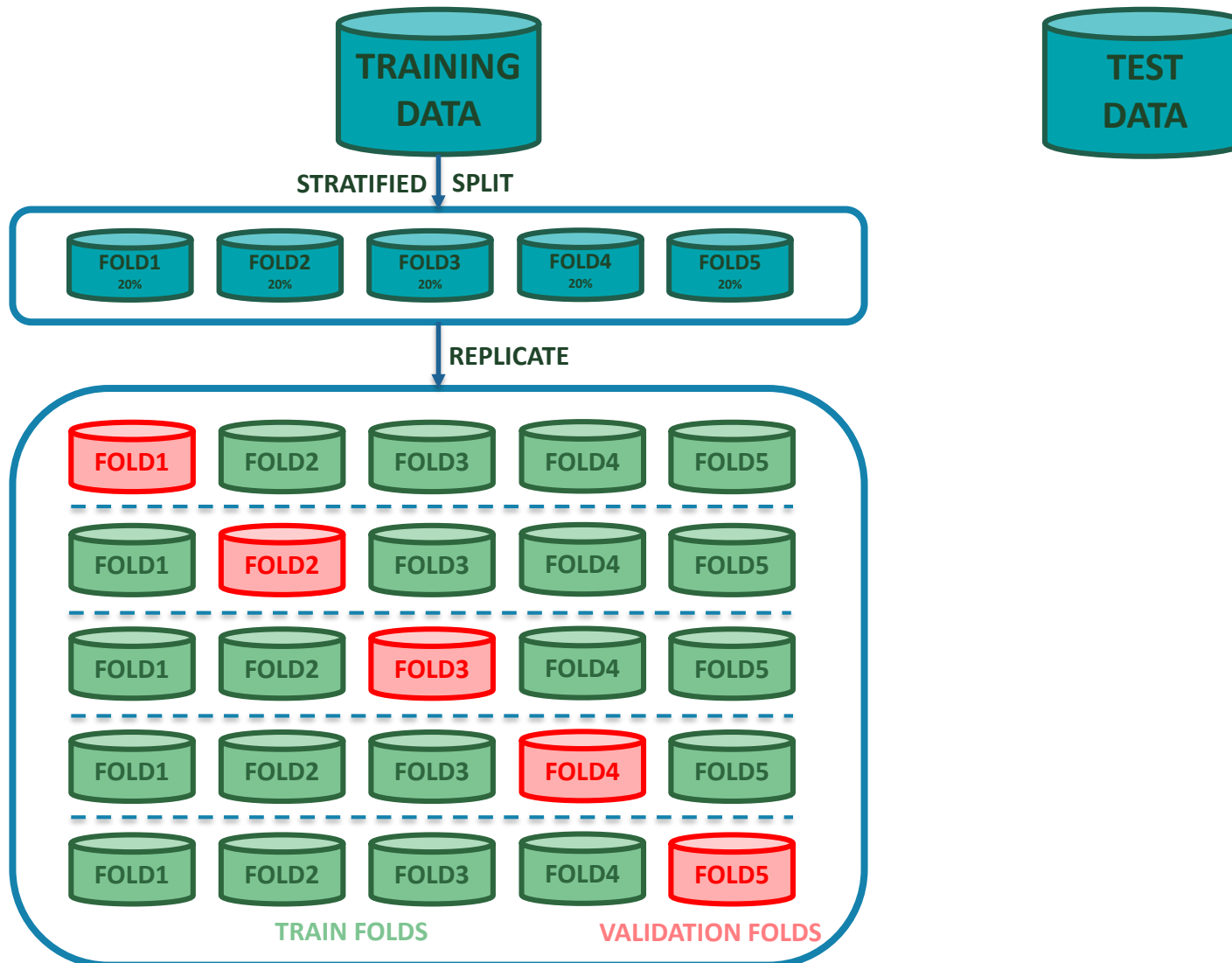
Train/Test vs Cross Validation



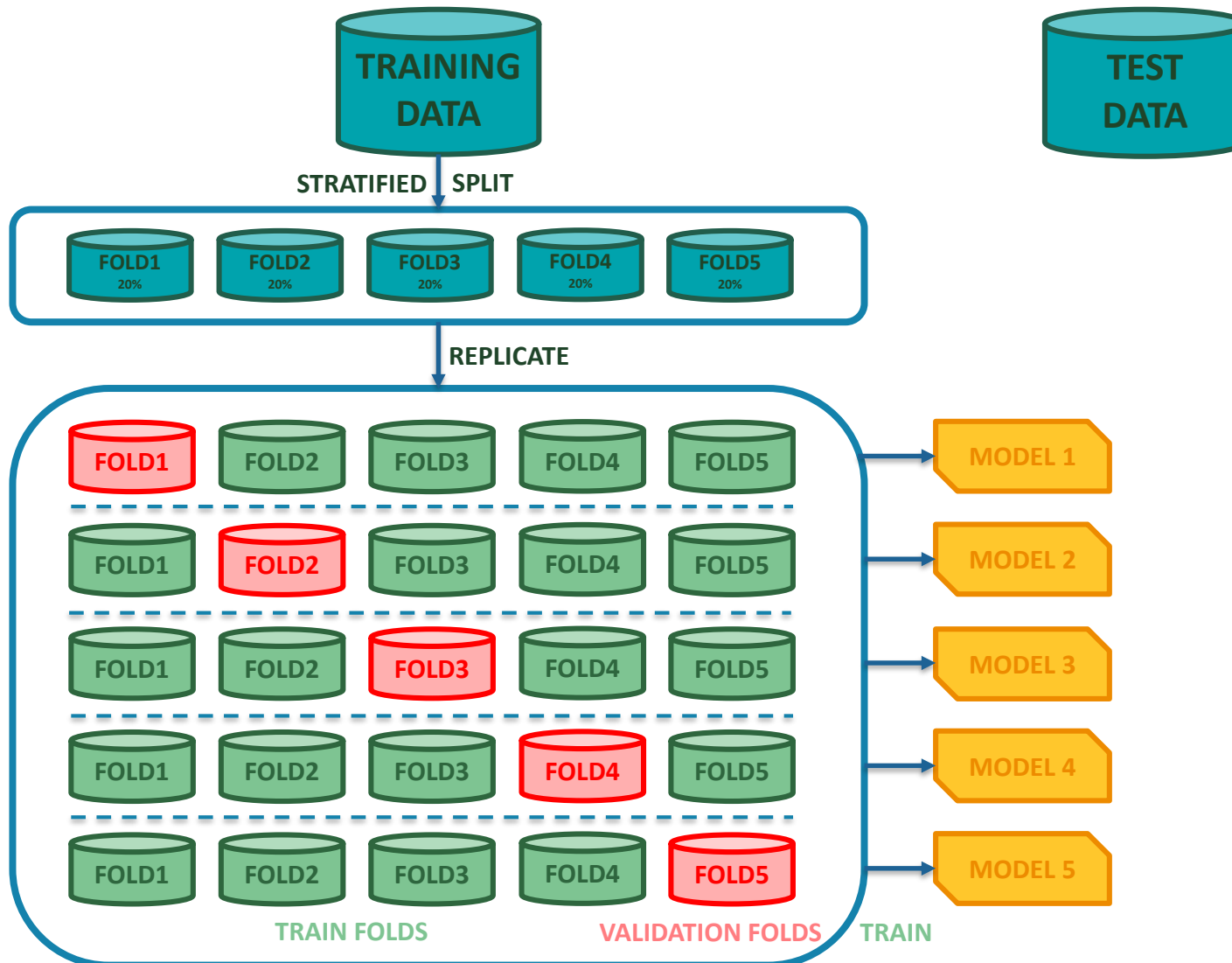
Train/Test vs Cross Validation



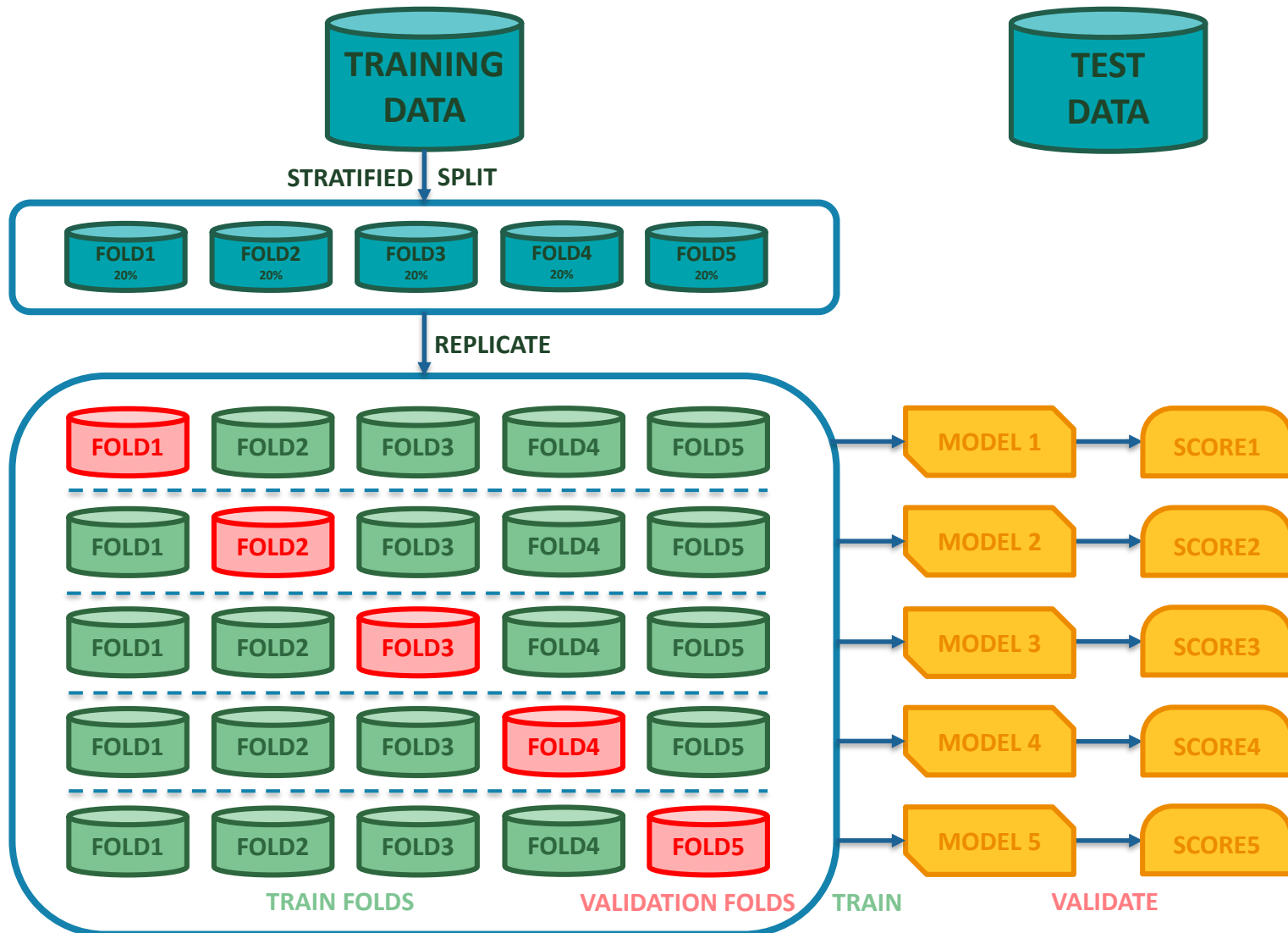
Train/Test vs Cross Validation



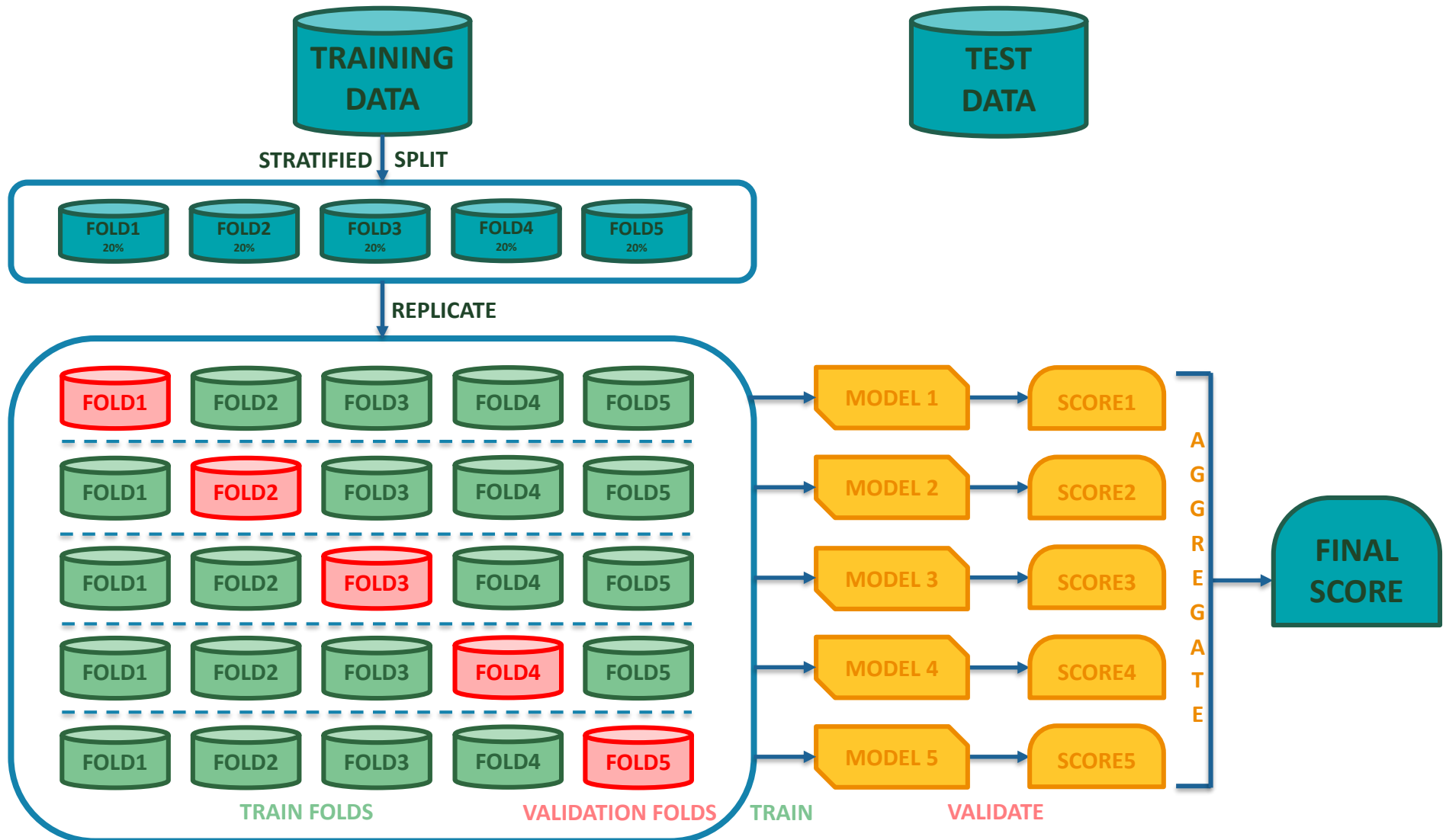
Train/Test vs Cross Validation



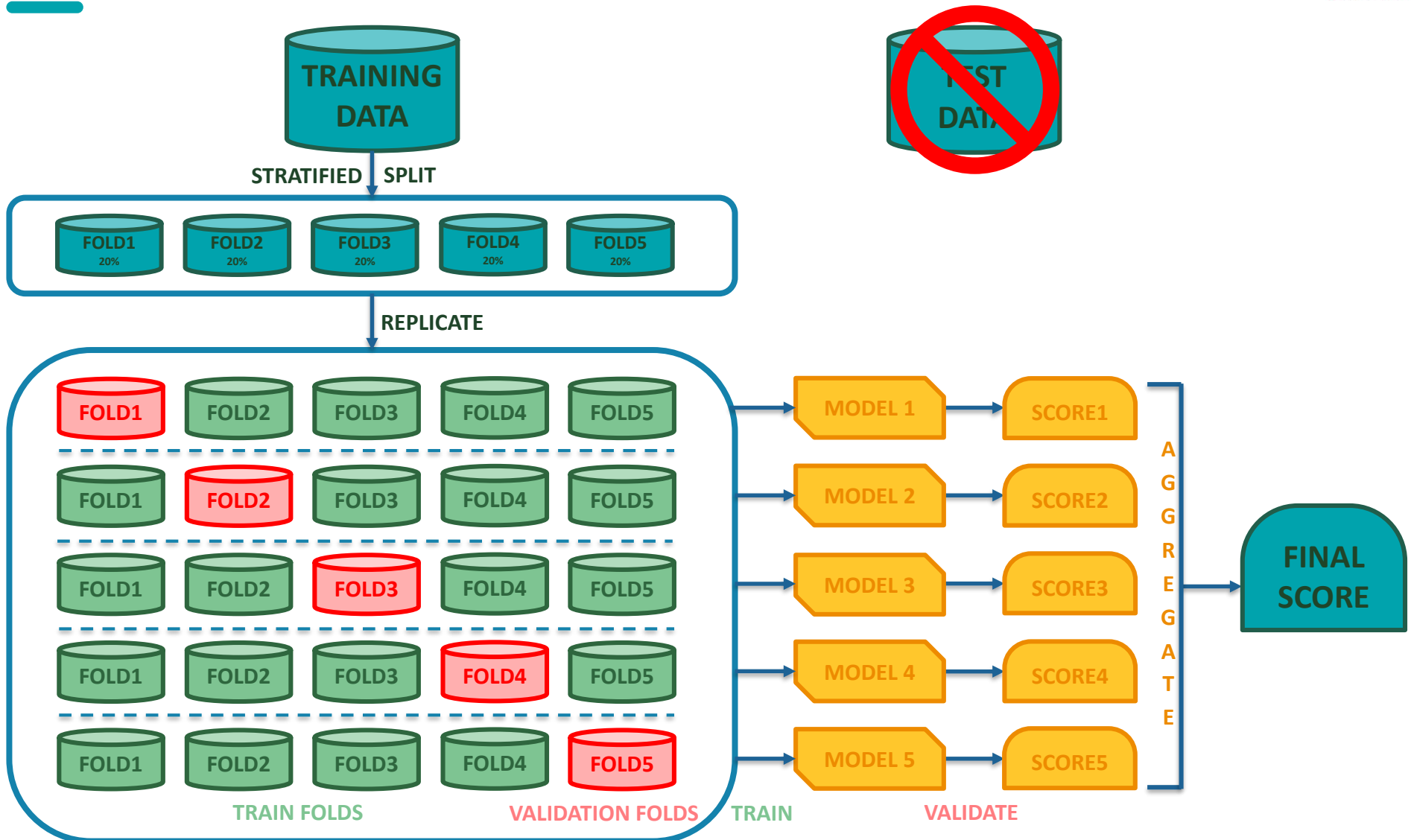
Train/Test vs Cross Validation



Train/Test vs Cross Validation



Train/Test vs Cross Validation



Cross-validation: K-fold

```
from sklearn.model_selection import KFold # importar KFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]]) # creamos un array
y = np.array([1, 2, 3, 4]) # creamos otro array
kf = KFold(n_splits=2) # Definimos el número de particiones
```

KFold(n_splits=2, random_state=None, shuffle=False)

Cross-validation: ventajas

1. Utilización de todos los datos
 - i. Cuando hay pocos datos: la división train / test puede dejarnos con pocos datos de test
 - a. Si el conjunto de test es pequeño: el resultado obtenido con el conjunto de test puede ser **fruto del azar**
 - b. Problemas multiclase: pocas muestras por clase
 - ii. Utilizando K modelos diferentes
 - i. Hacemos predicciones en todos **nuestros** datos

Cross-validation: ventajas

2. Creando y testeando el modelo con K modelos

- i. Aseguramos un mejor rendimiento
 - i. En una sola evaluación podríamos obtener un resultado causado por el azar o causado por el sesgo en el conjunto de test

Cross-validation: ventajas

3. Parameter tuning

- Ejemplo, buscar la mejor K:

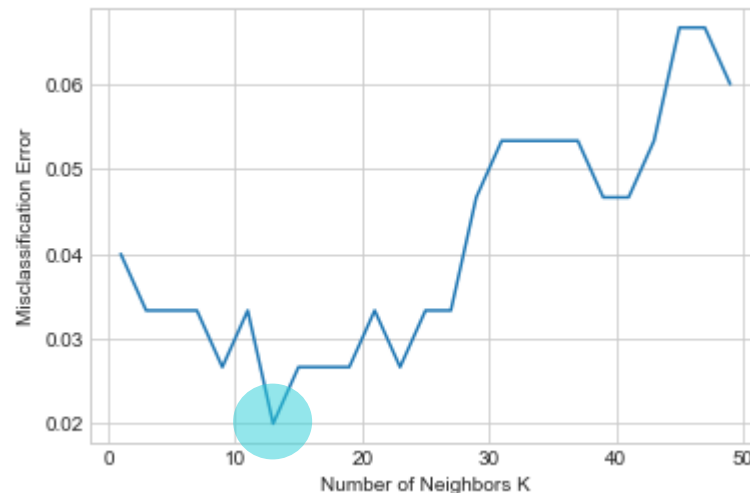
```
cv_scores = []
```

```
for k in neighbors:
```

```
    knn = KNeighborsClassifier(n_neighbors=k)
```

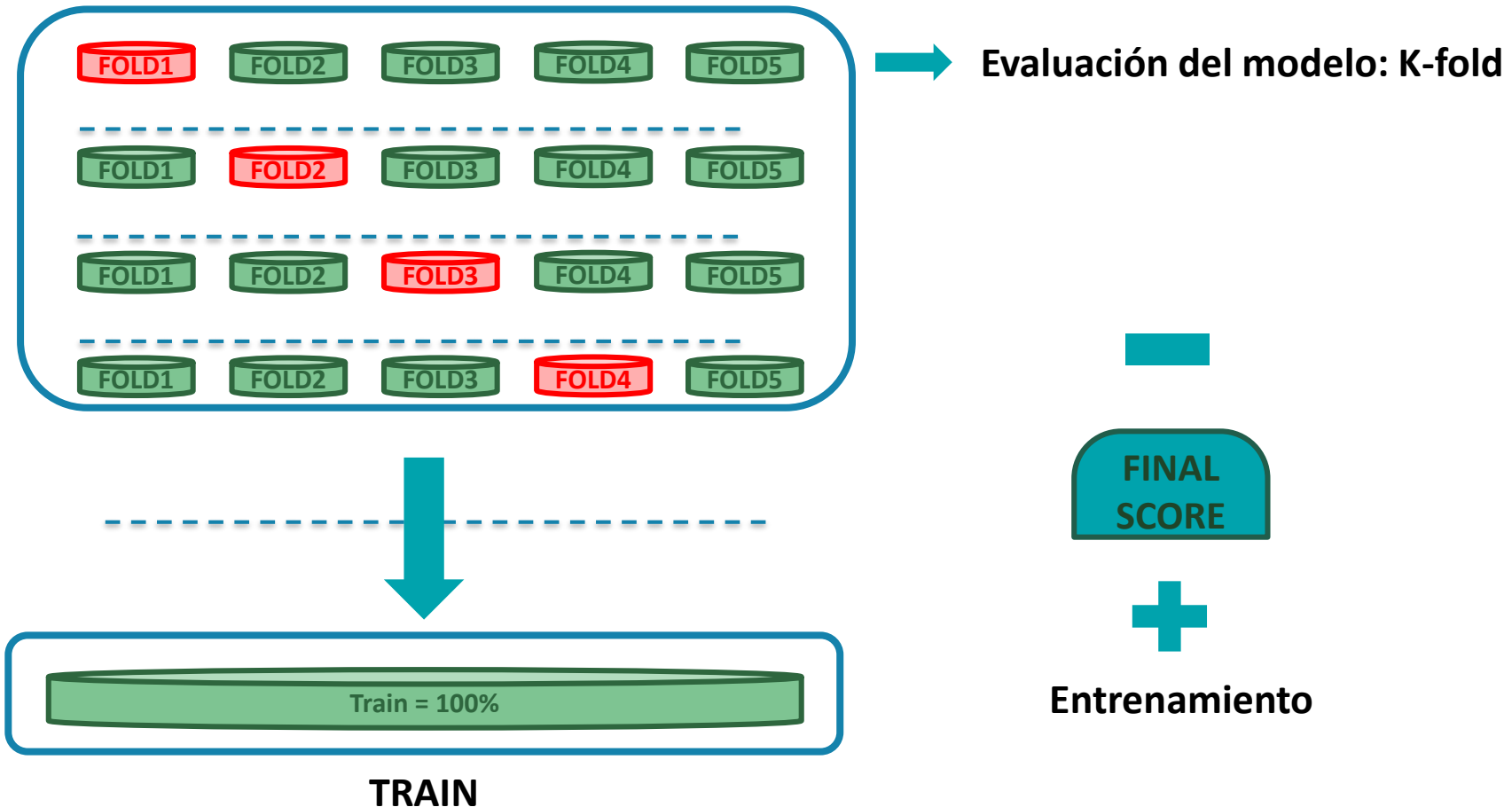
```
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
```

```
    cv_scores.append(scores.mean())
```



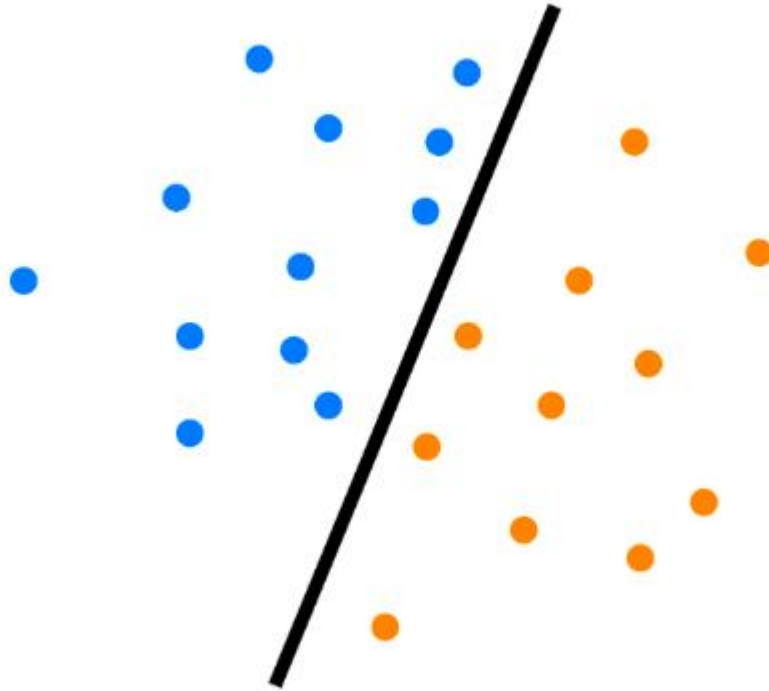
Cross-validation: desventajas

- Sesgo pesimista en resultado de rendimiento:
 - Normalmente el modelo mejorará con el entrenamiento con todo el conjunto



Cross-validation: estratificación

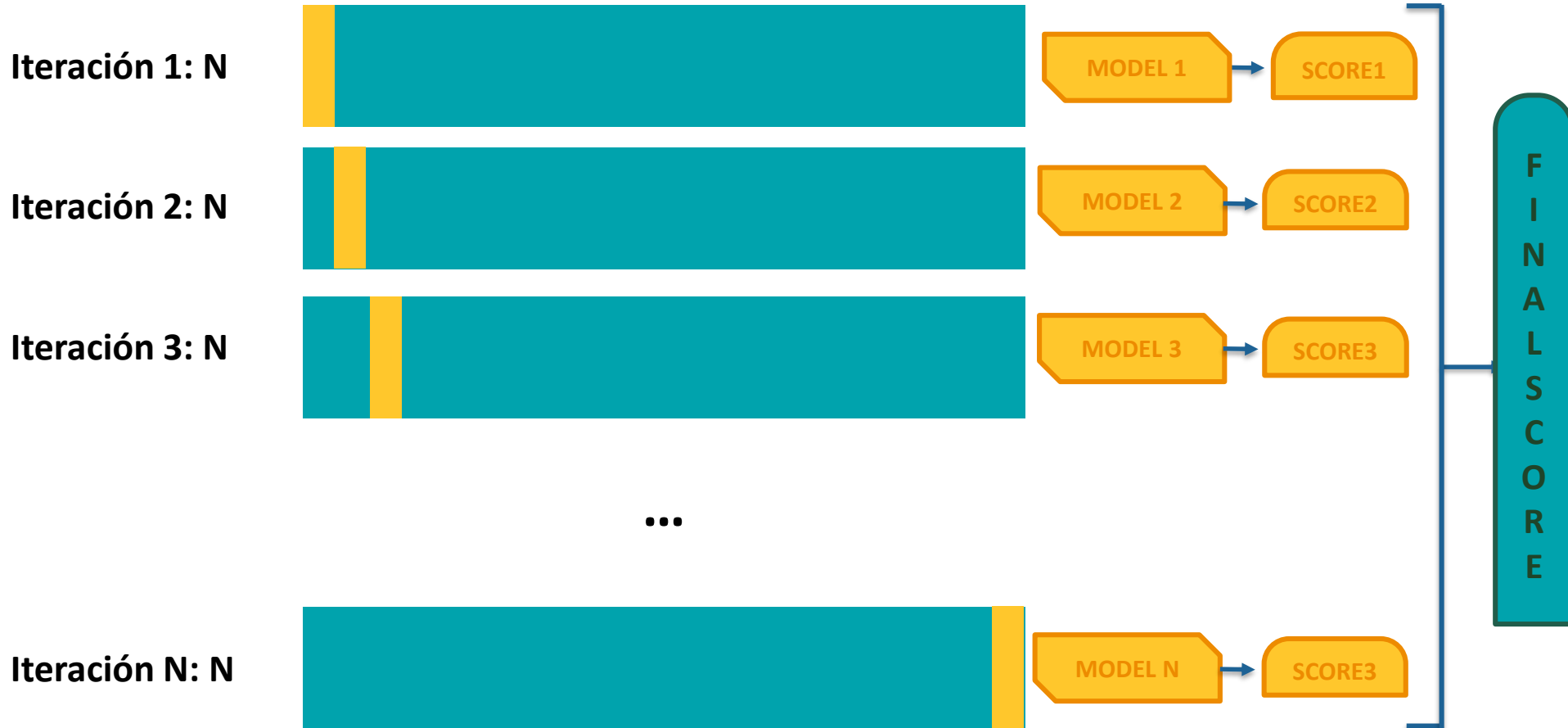
- Clasificación binaria / multiclase:
 - Estratificada por defecto en scikit-learn



Cross-validation: leave one out

- Se deja **1 punto** fuera de los datos de entrenamiento
 - Núm. de datos para **entrenamiento** $\Rightarrow N - 1$
 - **Validación**: en el punto dejado fuera
 - Se repite esto para todas las combinaciones donde la muestra original se puede separar
 - Resultado (score): media de cada iteración
 - Combinación total: número de datos total (n)

Cross-validation: leave one out



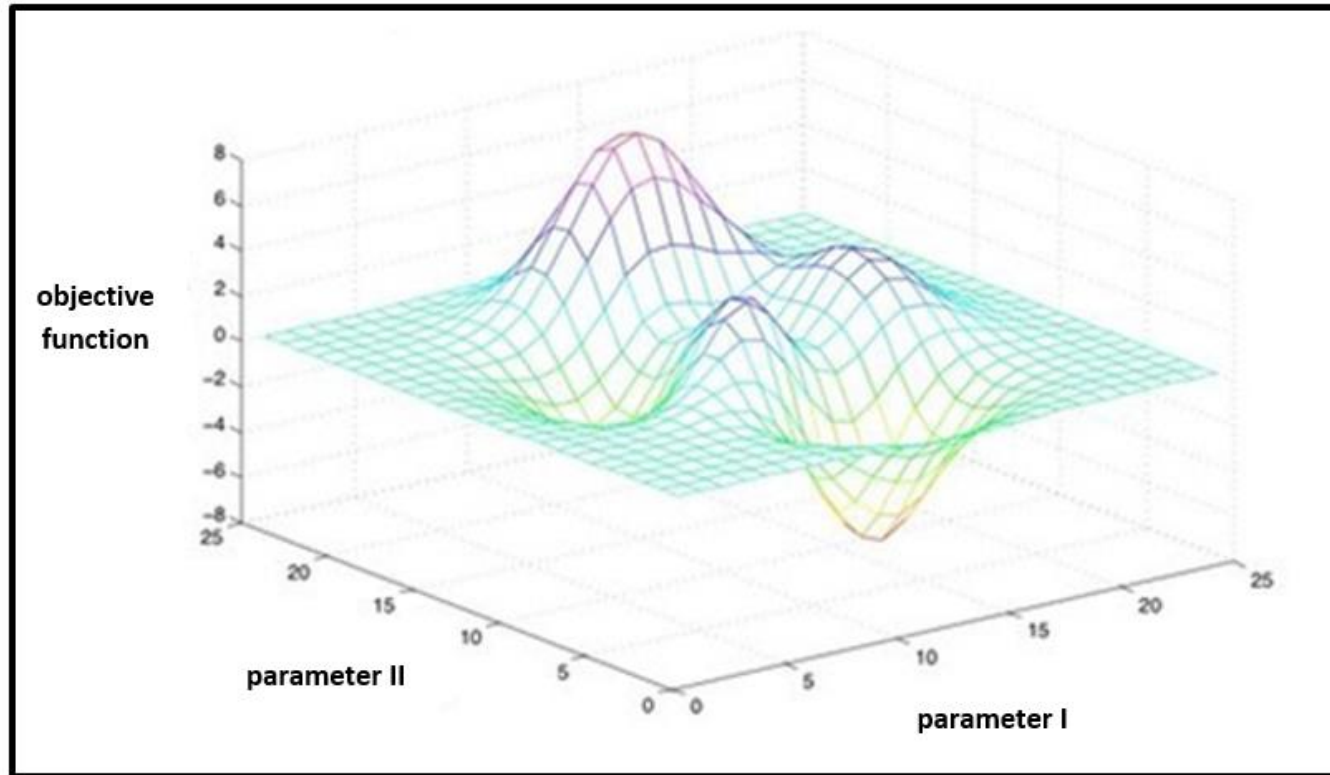
Cross-validation: leave one out

- **Ventajas:**
 - Sesgo reducido: la diferencia entre el conjunto de entrenamiento entero y el utilizado para cada fold es mínima
- **Desventajas:**
 - Costoso computacionalmente
 - Solapamiento entre conjuntos de entrenamiento: varianza



Grid search

- Una forma de “tunear” los hiperparámetros de un modelo
 - Objetivo: encontrar la mejor combinación



Parameter tuning

- Ejemplo de hiperparámetros:

```
class sklearn.neighbors. KNeighborsClassifier (n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,  
p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs) [source]
```

```
class sklearn.svm. SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True,  
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,  
decision_function_shape='ovr', random_state=None) [source]
```



entrenamiento del modelo

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=3) → Parámetros
```

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

predicciones en el conjunto de test

```
y_pred = knn.predict(X_test)
```

Parameter tuning

1. Manual:
 - Se eligen los hiperparámetros basándonos en la experiencia/intuición
2. Grid search
 - Establecemos un grid con hiperparámetros con todas las combinaciones
3. Random search
 - Grid con los valores de los hiperparámetros y seleccionamos combinaciones aleatorias
4. Automatizado
 - Se pueden utilizar métodos como gradient descent, Optimización Bayesiana o algoritmos evolutivos

Grid search

1. Por cada modelo:
 - Por cada combinación de parámetros
 - Cross-validation
 - Guardamos el score
2. Visualizamos y comparamos los resultados



**Mondragon
Unibertsitatea**

Goi Eskola
Politeknikoa

Aitor Agirre / Carlos Cernuda

aaguirre@mondragon.edu