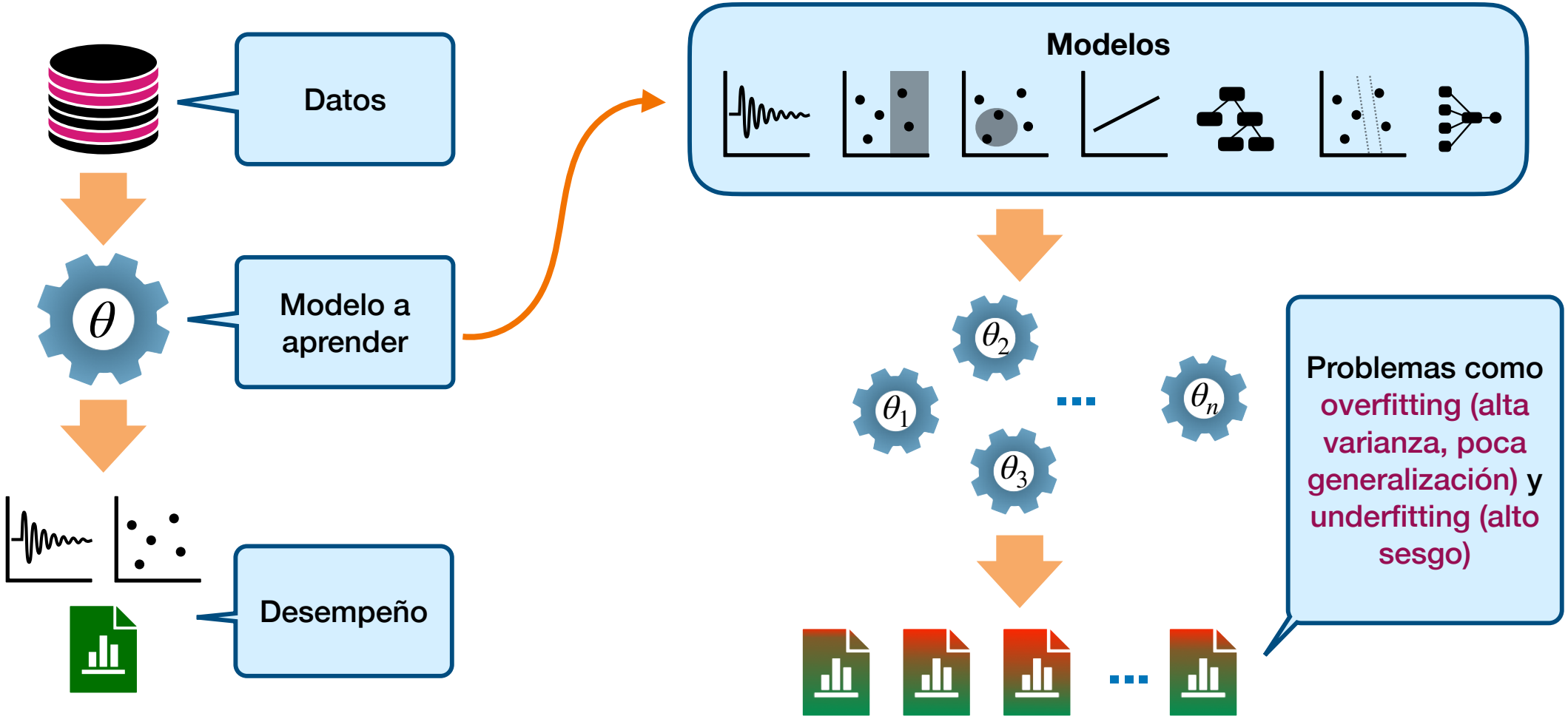
Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)

Aprendizaje Supervisado - Ensembles

ML Supervisado - Escenarios de ML



ML Supervisado - Ensembles

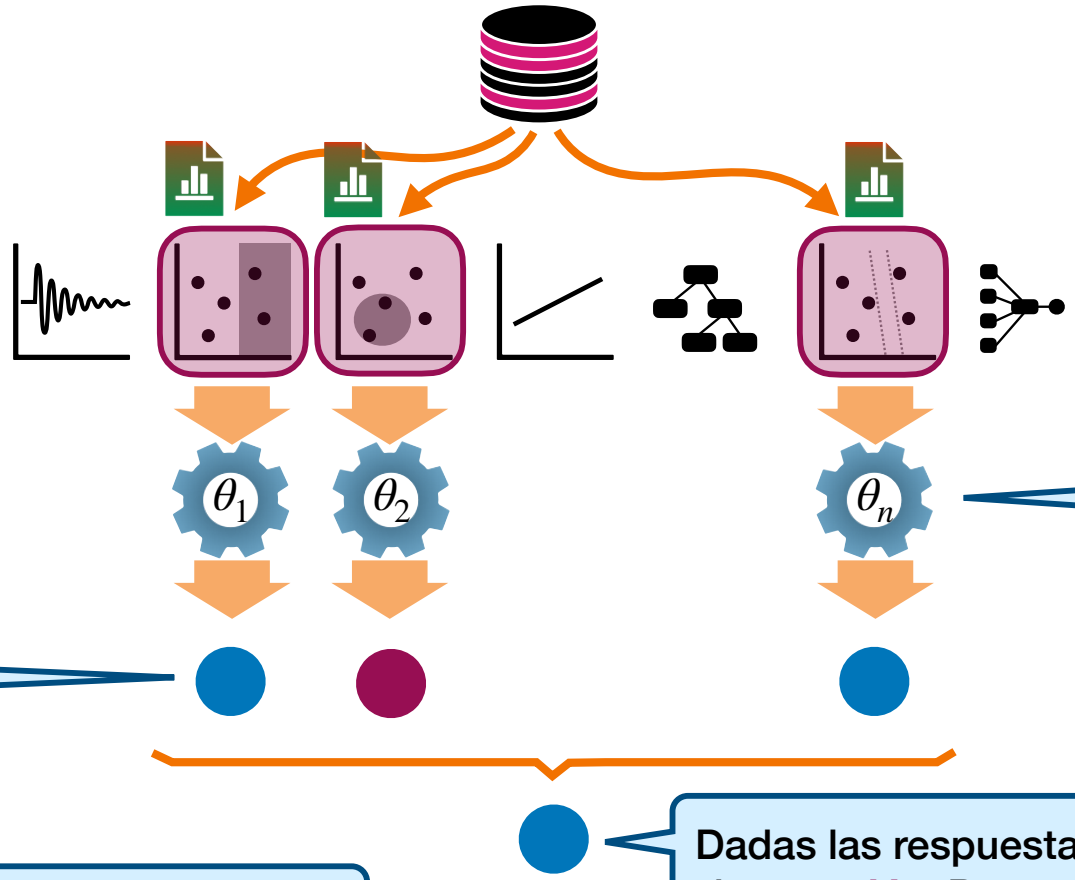
Un **ensemble** es una técnica en machine learning supervisado donde se combinan múltiples modelos para formar un modelo más robusto. Algunas de las técnicas de **ensembles** más conocidas son:

Bagging (Bootstrap Aggregating): Entrena varios modelos en paralelo sobre subconjuntos aleatorios de los datos. Luego combina sus predicciones (votación en clasificación, promedio en regresión). Ejemplo: **Random Forests** (construye muchos árboles de decisión).

Boosting: Entrena modelos secuencialmente, cada nuevo modelo corrige los errores de los anteriores. Se asignan más pesos a las instancias mal clasificadas. Ejemplos: AdaBoost, Gradient Boosting.

Stacking (Stacked Generalization): Combina diferentes tipos de modelos (heterogéneos). Usa un meta-modelo que aprende a integrar sus predicciones. Ejemplo: usar k-NN + SVM + árbol de decisión, y luego una regresión logística para combinar.

ML Supervisado - Ensembles - Voting



Se seleccionan diferentes tipos de modelos con buen desempeño

Se entrenan cada uno de los correspondientes modelos

Dada una consulta, se realiza la misma en cada modelo entrenado

Dadas las respuestas se somete a un proceso de ***votación***. Para problemas de **clasificación** se utiliza **mayority**, para uno de **regresión**, generalmente el **promedio** de los resultados

También se puede utilizar una **soft voting** utilizando probabilidades

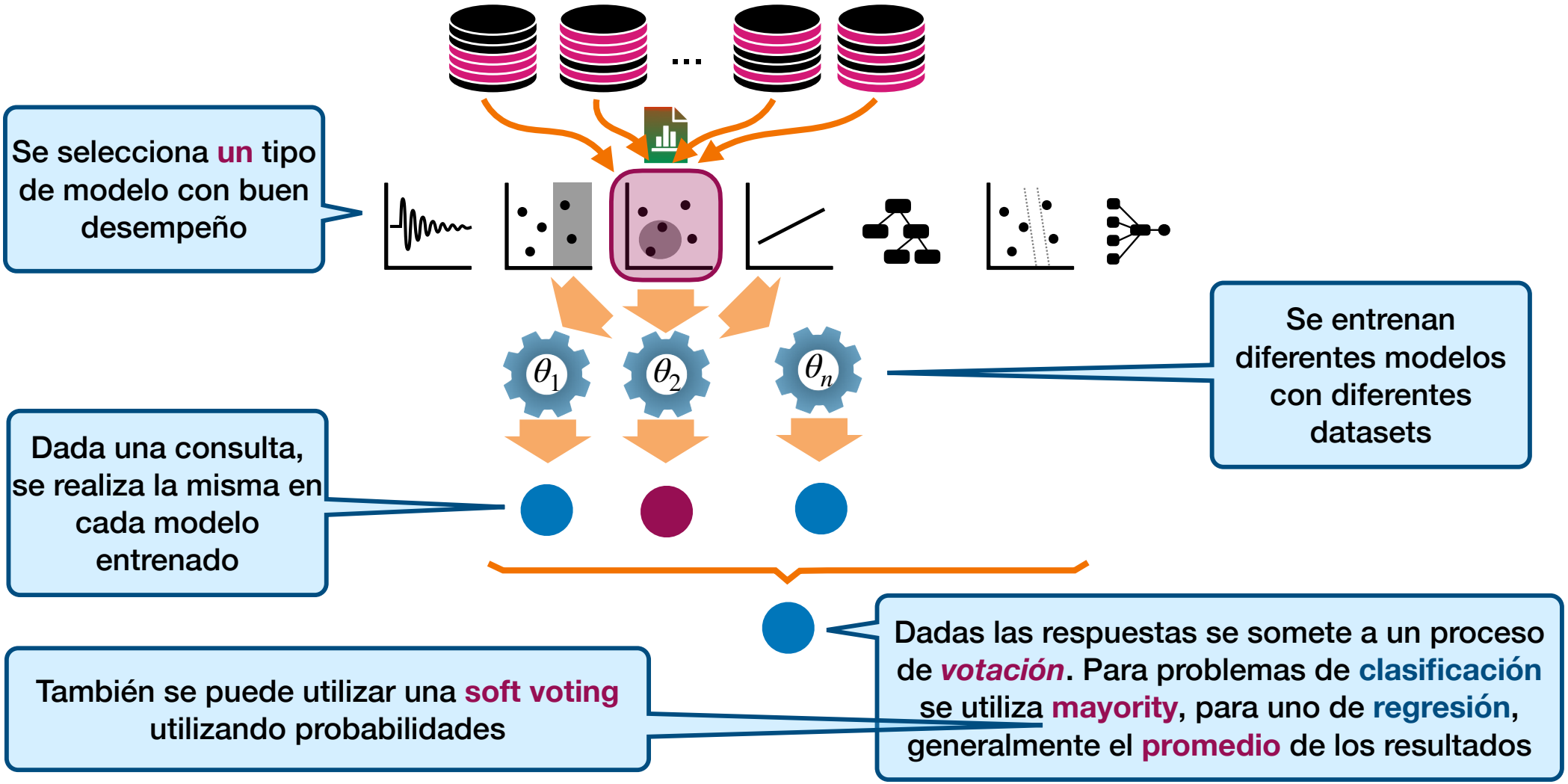
MLS Clasificación - Voting con ScikitLearn

```python
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
...

# Datos
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Modelos base
log_clf = LogisticRegression(max_iter=500, random_state=42)
svm_clf = SVC(probability=True, random_state=42)
tree_clf = DecisionTreeClassifier(random_state=42)

# Voting Classifier con soft voting
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('svc', svm_clf), ('dt', tree_clf)],
    voting='soft'
)
voting_clf.fit(X_train, y_train)

# Predicciones con probabilidades
probas = voting_clf.predict_proba(X_test[:1])  # testeamos con una muestra
print("Probabilidades promedio (soft voting):", probas)
print("Predicción final:", voting_clf.predict(X_test[:1]))
```

Configuración de los tipos de modelos elegidos

Configuración de Soft Voting (con probabilidad)

ML Supervisado - Ensembles - Bagging



MLS Clasificación - Bagging con ScikitLearn

Configuración de tipo de modelo elegido

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
...

# Cargar el dataset Iris
X, y = load_iris(return_X_y=True)
# Dividir en entrenamiento y test (70%-30%)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Bagging con árboles de decisión
bagging_clf = BaggingClassifier(
    estimator=DecisionTreeClassifier(),  # modelo base
    n_estimators=50,       # número de clasificadores
    max_samples=0.8,       # % de muestras usadas en cada bootstrap
    bootstrap=True,        # habilita bootstrap sampling
    random_state=42
)

bagging_clf.fit(X_train, y_train)
y_pred = bagging_clf.predict(X_test)
print("Accuracy Bagging:", accuracy_score(y_test, y_pred))
```

MLS Clasificación - Bagging - Random Forest

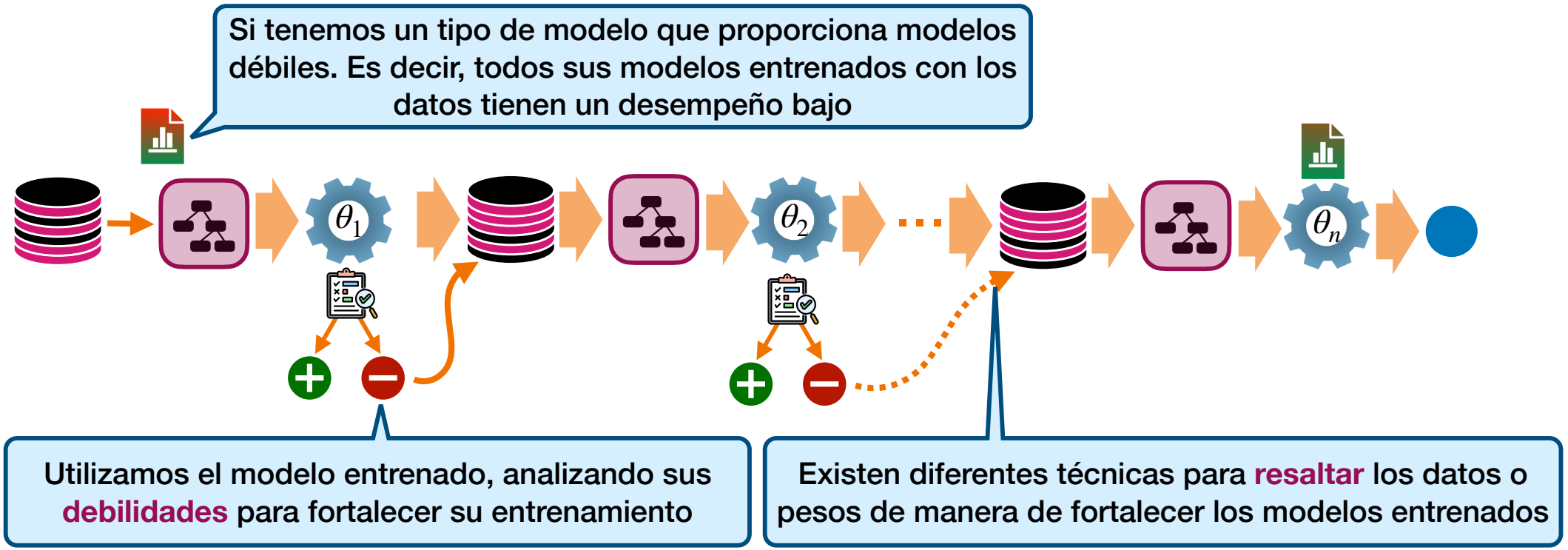
Random Forest es un algoritmo de **ensemble** basado en **bagging** que combina múltiples árboles de decisión para mejorar la precisión y la capacidad de generalización. Fue propuesto por Leo Breiman en 2001 como una extensión del bagging, con **la innovación de agregar aleatoriedad en la selección de atributos en cada división del árbol**.

Porcentaje de la muestra original que utiliza para cada árbol

```python
from sklearn.ensemble import RandomForestClassifier


rf_clf = RandomForestClassifier(
    n_estimators=100,
    max_samples=0.8,
    random_state=42
)
```

MLS Clasificación - Boosting

Boosting es una técnica que combina **múltiples modelos débiles (weak learners)** para formar un **modelo fuerte (strong learner)**. La idea es entrenar de **manera secuencial** a varios modelos, donde cada modelo nuevo se enfoca en corregir los errores cometidos por los anteriores. En lugar de entrenar todos los modelos en paralelo (como en bagging), los modelos se construyen uno tras otro ajustando la **importancia de las instancias según la dificultad de clasificarlas**.



MLS Clasificación - Boosting

AdaBoost (Adaptive Boosting):
- Primer algoritmo popular de boosting.
- Ajusta pesos en las muestras mal clasificadas.

Gradient Boosting:
- En lugar de ajustar pesos directamente, cada nuevo modelo se entrena para reducir el error (gradiente) del modelo anterior.
- Muy flexible, usado en regresión y clasificación.

XGBoost (Extreme Gradient Boosting):
- Optimización de Gradient Boosting con técnicas para mayor velocidad y regularización.

LightGBM y CatBoost:
- Variantes modernas enfocadas en datasets grandes, con optimizaciones en memoria y tiempo de entrenamiento.

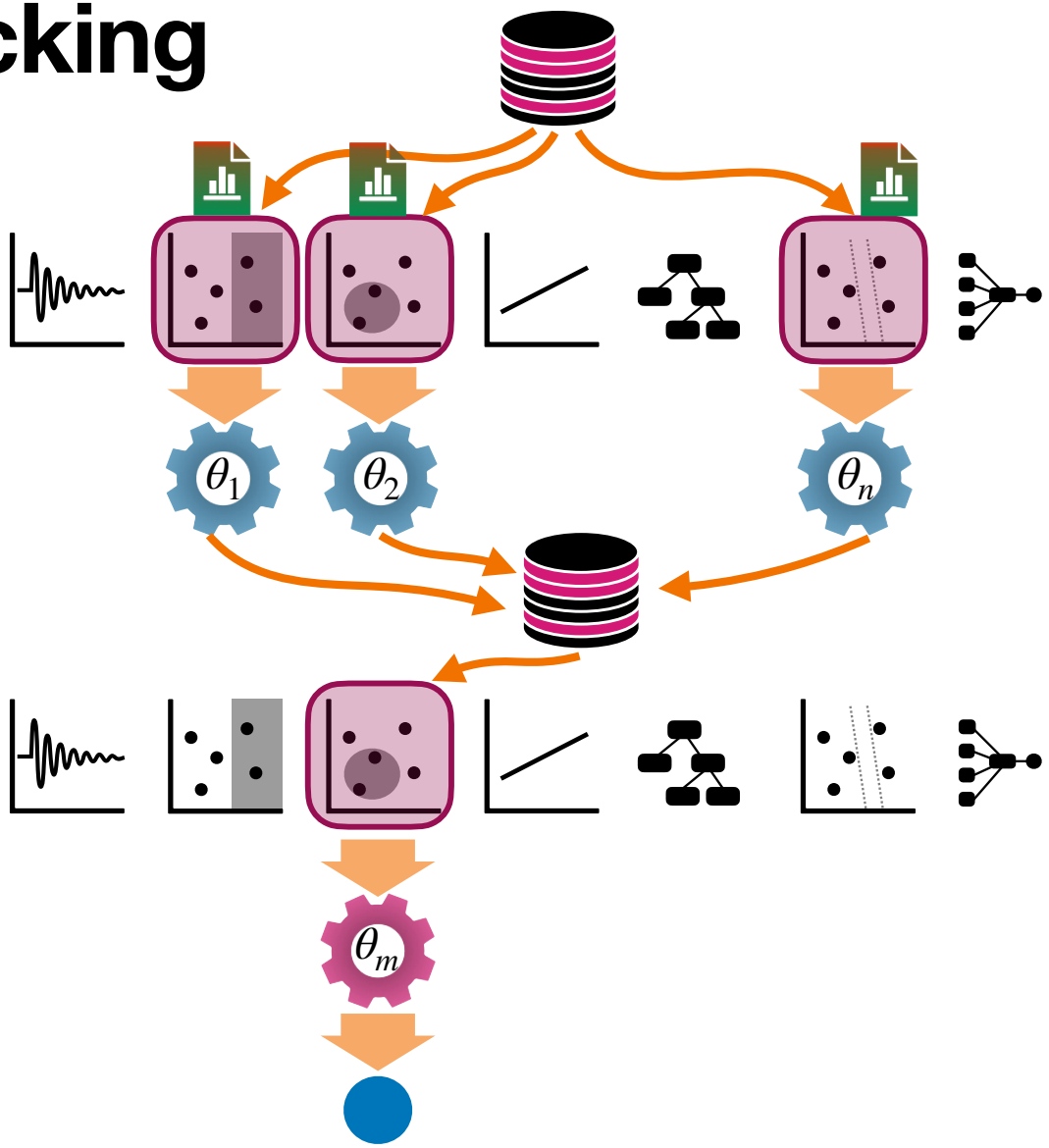
MLS Clasificación - Boosting con ScikitLearn

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
...

# Cargar y dividir el dataset Iris
X, y = load_iris(return_X_y=True)
train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Modelo base (árbol de decisión poco profundo = weak learner)
base_tree = DecisionTreeClassifier(max_depth=1, random_state=42)

# Crear el modelo AdaBoost
ada_clf = AdaBoostClassifier(
    estimator=base_tree,
    n_estimators=50,        # cantidad de modelos débiles
    learning_rate=1.0,      # controla el peso de cada clasificador
    random_state=42
)

ada_clf.fit(X_train, y_train)
y_pred = ada_clf.predict(X_test)
print("Accuracy con AdaBoost:", accuracy_score(y_test, y_pred))
```

MLS Clasificación - Stacking

Stacking (Stacked Generalization) es una técnica donde se combinan varios modelos (llamados **modelos base**) y un modelo adicional (**meta-modelo**) que **aprende a combinar sus predicciones**.

A diferencia de Bagging o Boosting, que combinan modelos de la misma familia, por ejemplo, muchos DT, Stacking puede **mezclar modelos heterogéneos**: árboles, regresiones, redes neuronales, SVM, etc.



MLS Clasificación - Stacking

```
from sklearn.ensemble import StackingClassifier
...
# Datos
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Modelos base
base_estimators = [
    ('dt', DecisionTreeClassifier(max_depth=5)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
]
# Meta-modelo
meta_model = LogisticRegression()
# Stacking
stack_model = StackingClassifier(
    estimators=base_estimators,
    final_estimator=meta_model,
    cv=5 #k-fold cross-validation
)
stack_model.fit(X_train, y_train)
y_pred = stack_model.predict(X_test)
```

MLS Clasificación - Ensembles - Comparación

Característica	Bagging	Boosting	Stacking
Idea principal	Entrenar varios modelos iguales en subconjuntos aleatorios de los datos y promediar	Entrenar modelos secuenciales corrigiendo errores del anterior.	Combinar predicciones de distintos modelos base mediante un modelo “meta”.
Objetivo	Reducir varianza y evitar overfitting.	Reducir bias y mejorar precisión.	Mejorar desempeño global combinando modelos heterogéneos.
Cómo combina modelos	Votación (clasificación) o promedio (regresión).	Ponderación de modelos según desempeño; secuencial.	Modelo meta aprende a partir de las predicciones de los modelos base.
Ejemplos comunes	Random Forest, Bagged Decision Trees	AdaBoost, Gradient Boosting, XGBoost	Stacking Classifier/Regressor
Ventajas	Robusto a ruido, fácil paralelizar.	Alta precisión, corrige errores de modelos débiles.	Flexible, aprovecha lo mejor de cada modelo.
Desventajas	Puede no reducir bias, menos eficiente con modelos débiles.	Sensible a ruido y outliers, más lento.	Más complejo de implementar, requiere buen modelo meta.