

Tema 04. Diseño de sistemas de Aprendizaje Automático

Autor: Ismael Sagredo Olivenza

4.1 Proceso de desarrollo del ML

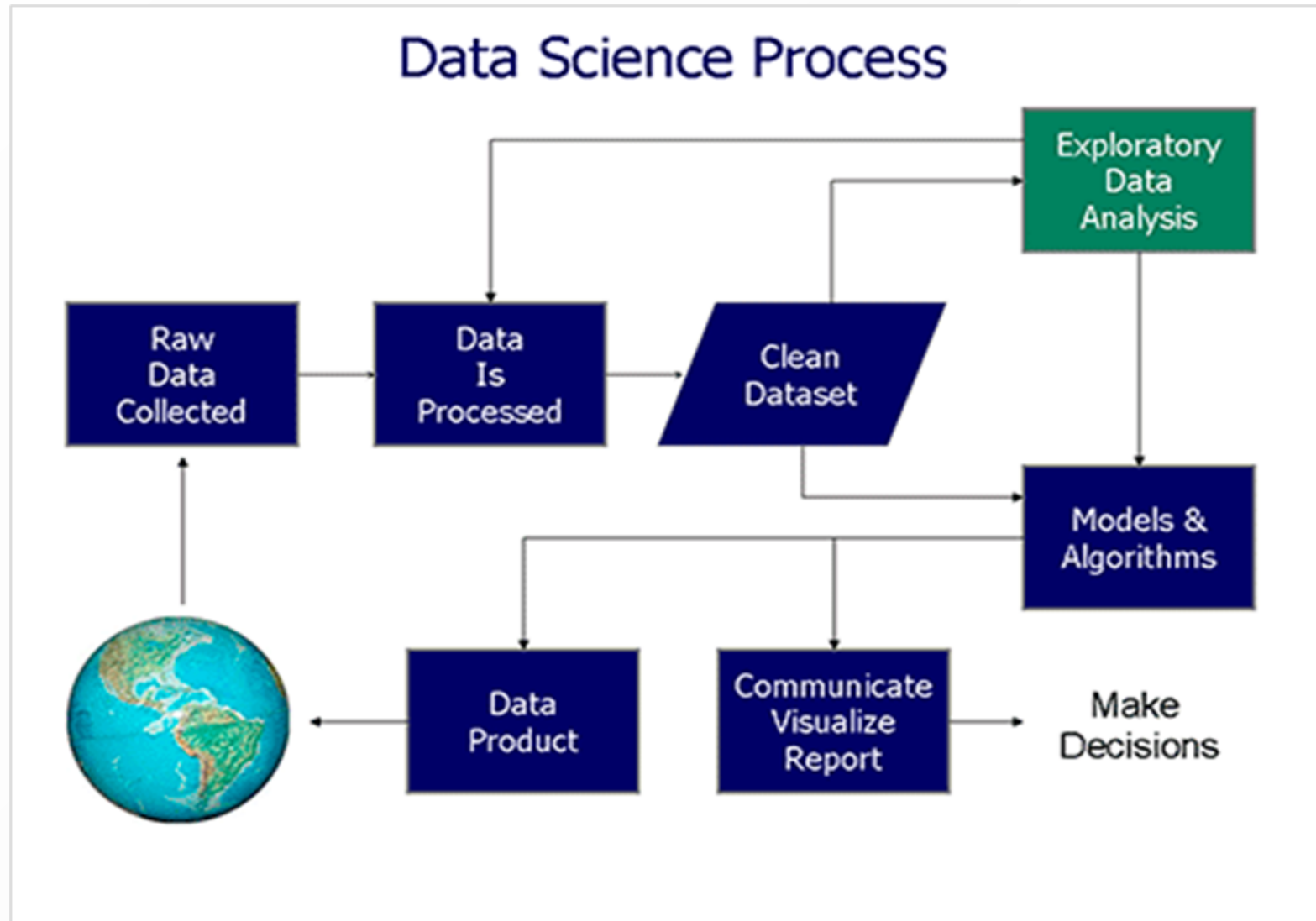
En proyectos reales:

Gran parte del tiempo de desarrollo se dedica a obtener, pulir y limpiar datos.

Los algoritmos de ML suelen tener que volver a entrenarse para mantener su relevancia, ya que el entorno cambia a menudo. Y durante el entrenamiento hay que hacer muchos ajustes finos.

Así, tenemos un proceso iterativo de desarrollo de proyectos basados en aprendizaje automático.

4.1.1 Data Science Process

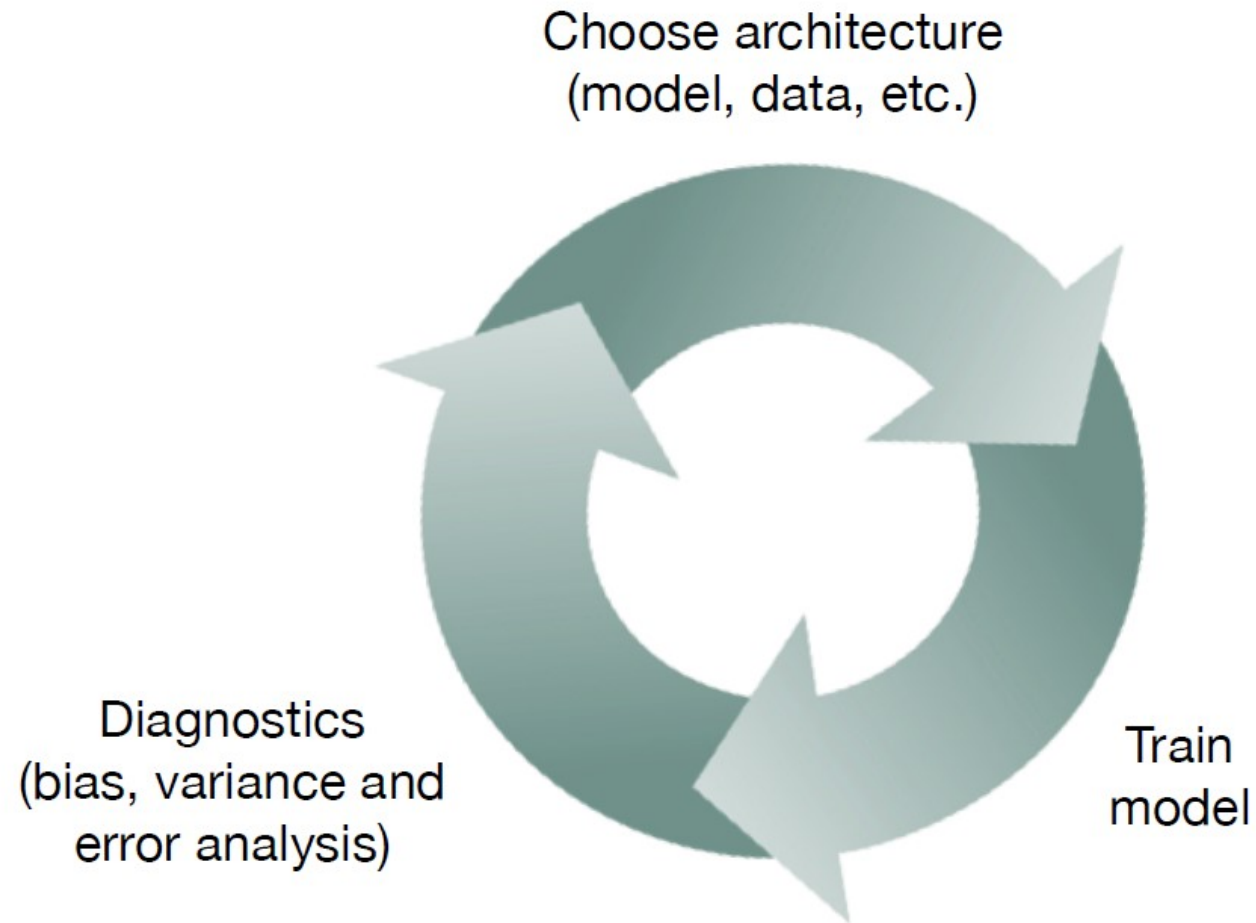


Este proceso iterativo se refleja en el **Proceso de Ciencia de Datos**. En muchas ocasiones, la modificación del mundo por parte de la propia IA hace que el entorno cambie, por lo que debemos repetir el proceso de nuevo.

Si nos centramos únicamente en el proceso iterativo desde el punto de vista del Machine Learning, podemos simplificar el proceso a tres fases:

- Elegir la arquitectura: incluyendo datos, modelos, etc.
- Entrenar el modelo: Entrenamiento del propio modelo, selección de hiperparámetros, etc.
- Diagnóstico: sesgo, varianza, error, etc.

4.1.2 Iterative loop of ML development



4.2 Limpiar y codificar los datos

Limpiar el conjunto de datos de errores y valores nulos. Estrategias:

- Eliminar observaciones: Si hay pocos datos no es recomendable.
- Imputar al campo o característica un valor que tenga sentido según los datos disponibles. Por ejemplo el valor medio / moda

```
from sklearn.impute import SimpleImputer
```

- Imputar el valor medio de las entradas más similares al dato que estamos inputando (KNNImputer)

```
from sklearn.impute import KNNImputer
```

4.2.1 One-hot encoding

Al trabajar con variables categóricas, una de las cuestiones más importantes es transformar nuestras variables categóricas en variables numéricas.

One-Hot Encoding transforma una característica categórica en tantas variables como valores posibles tenga la categoría y asigna a cada una de ellas 0 ó 1 en función del valor.

```
from sklearn.preprocessing import OneHotEncoder
oneHotEncoder = OneHotEncoder(drop='first')
oneHotEncoderFit = oneHotEncoder.fit(provincias.reshape(-1,1))
oneHotEncoderFit.transform(provincias.reshape(-1,1)).toarray()
```

4.3 Evaluating the model

- **Mean Square Error** entre el pronóstico y el real.

$$MSE(Y, Y') = \frac{1}{N} \cdot \sum_{i=1}^N (y_i - y'_i)^2$$

- **Confussion Matrix** (Ejemplo para clase binaria)

	C observed T	C observed F
C Predicted T	True Positive (TP)	False Positive (FP)
C Predicted F	False Negative (FN)	True Negative (TN)

- **Accuracy** o tasa de exactitud: Aciertos entre el total.

$$Accuracy = \frac{TP + TN}{TP + N + FP + FN} = \frac{V}{N}$$

La exactitud puede presentar problemas en los casos en que las clases estén desequilibradas.

- True Positive Rate / **Sensitivity (Recall)**. Verdaderos positivos entre todos los positivos observados

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

- Predictive Positive Value (**Precision**). True positives among all the positives predicted

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

- **F1 Score** $F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$

		Actual Outcome		
		Positive	Negative	
Predicted Outcome	Positive	True Positive	False Positive	Positive Predicted Value = TP / (TP + FP)
	Negative	False Negative	True Negative	

		Actual Outcome		
		Positive	Negative	
Predicted Outcome	Positive	True Positive	False Positive	Sensitivity = TP / (TP+FN)
	Negative	False Negative	True Negative	

Ejemplo

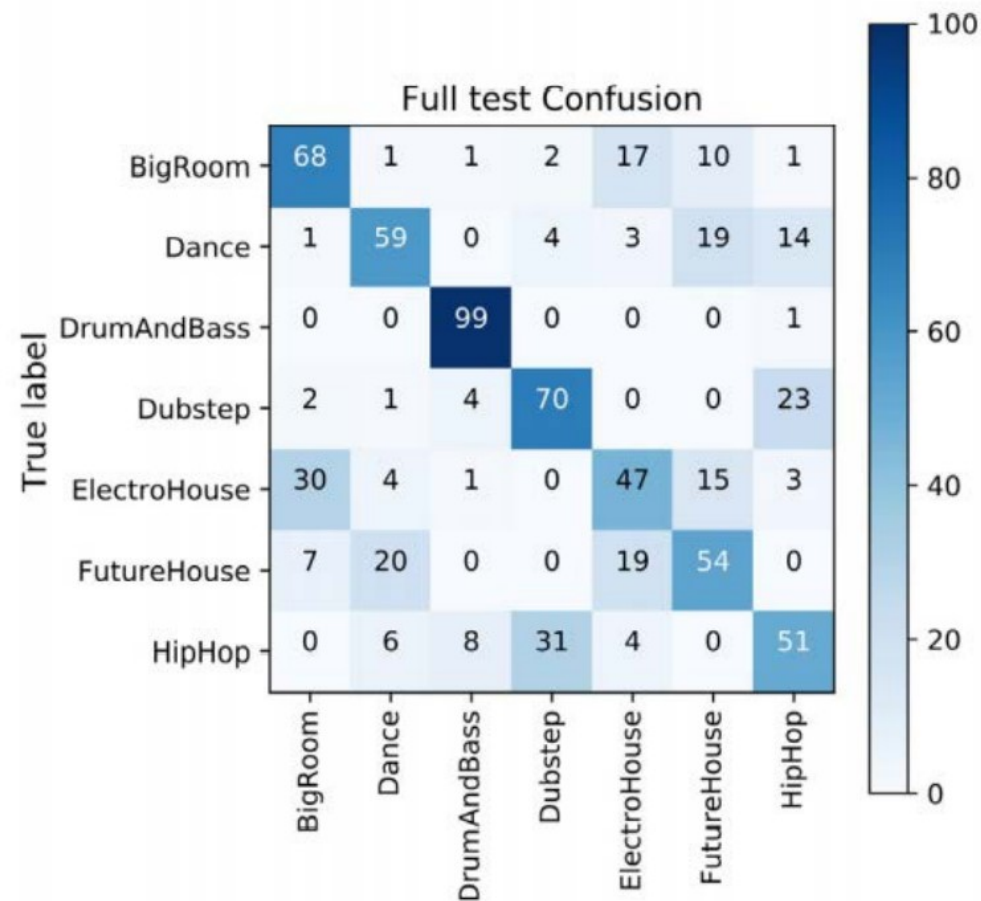
Confussion Matrix: Calculate precission, recall, F1Score.

	C observed T	C observed F
C Predicted T	176 (TP)	107 (FP)
C Predicted F	316 (FN)	993 (TN)

- Precision: $precision = \frac{TP}{TP+FP} = \frac{176}{176+107} = 0,6219 = 62,19\%$
- Recall: $recall = \frac{TP}{TP+FN} = \frac{176}{176+316} = 0,3577 = 35,77\%$
- F1Score: $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} = 2 \cdot \frac{0.62 \cdot 0.35}{0.62+0.35} = 0,4488$

Ejercicio

Interpreta la siguiente matriz de confusión. Clasificar canciones según el subgénero de música electrónica al que pertenecen



4.3.1 Testing

Dividir el conjunto de datos en datos de entrenamiento y datos de prueba.

La distribución de los datos entre prueba y entrenamiento debe ser correcta y proporcional al número de clases. Una distribución sesgada (Biases) puede causar problemas.

SKLearn nos puede ayudar en esta tarea:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(data, y, train_size = 0.8, random_state = 1234)
```

Durante la fase de entrenamiento, no sólo debemos fijarnos en el error de entrenamiento, sino también en el **error de validación**. El error de validación sería similar al **error de prueba**, pero se ejecuta durante el entrenamiento para ver si el modelo está perdiendo capacidad de generalización.

A priori podría parecer que la validación y la prueba son lo mismo, pero estos dos conjuntos se suelen diferenciar porque cuando se prueban diferentes ajustes del modelo (hiperparámetros) existe **aún un riesgo de sobreajuste** en el **conjunto de prueba** porque los parámetros se pueden **ajustar hasta que el estimador funcione de forma óptima**. De este modo, el conocimiento sobre el conjunto de pruebas puede "filtrarse" en el modelo y las métricas de evaluación ya no informan sobre el rendimiento de la capacidad de generalización.

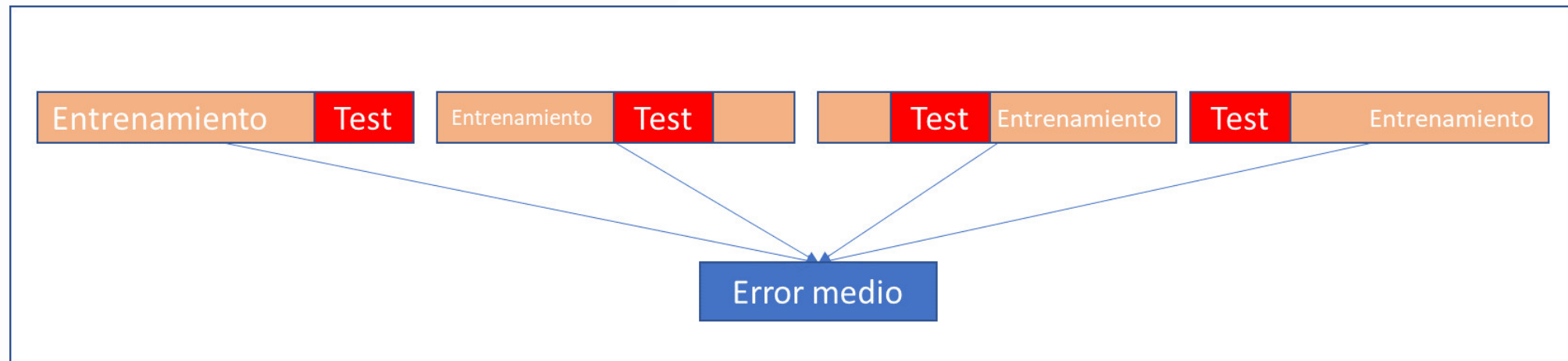
Por lo tanto, tendremos los siguientes conjuntos:

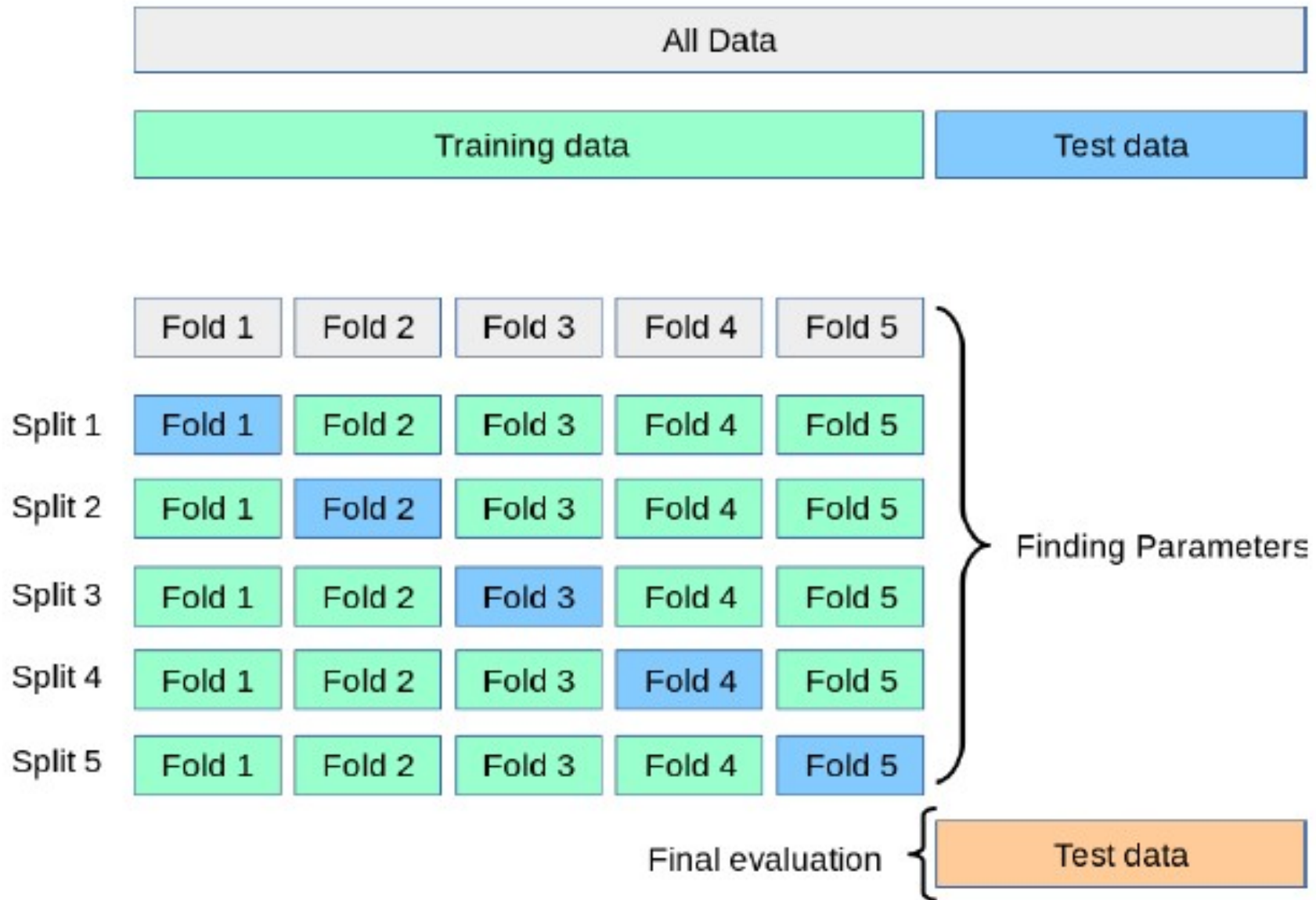
- **Error de entrenamiento:** Error de datos de entrenamiento.
- **Error de validación** (validation): Error de validación durante el entrenamiento para mantener la capacidad de generalización.
- **Error de pruebas** (test): Error con datos completamente excluidos del entrenamiento que tiene lugar después del entrenamiento.

Sin embargo, al dividir los datos disponibles en tres conjuntos, reducimos drásticamente el número de muestras que pueden utilizarse para el aprendizaje del modelo, y los resultados pueden depender de una elección aleatoria concreta para el par de conjuntos (entrenamiento, validación). Una solución es la **validación cruzada** o **cross-validation**

K-Fold Cross Validation

Cross-validation nos permite dividir los datos en k subgrupos y utilizar para el entrenamiento $k-1$ de ellos y para la validación durante el entrenamiento el restante. El error final es el error medio. Esto nos permite aprovechar los datos de entrenamiento y evitar posibles sesgos en la elección del conjunto de validación.





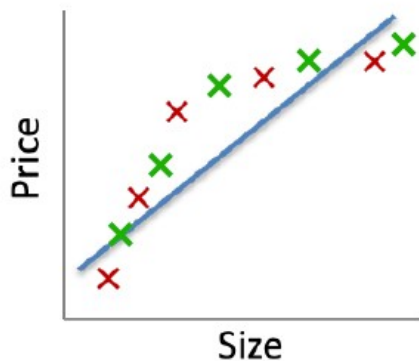
```
from sklearn.model_selection import cross_val_score
clf = Stimator(For example MLPClassifier)
scores = cross_val_score(clf, X, y, cv=5)
scores
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

Por defecto, la puntuación calculada en cada iteración de CV es el método de puntuación del estimador. Es posible cambiar esto utilizando el parámetro de puntuación:

```
scores = cross_val_score(clf, X, y, cv=5, scoring='f1_macro')
```

- Leave One Out (LOO)
- Leave P Out (LPO), Leave P Out (LPO)
- ShuffleSplit...

4.4 Diagnostic Bias and Variance



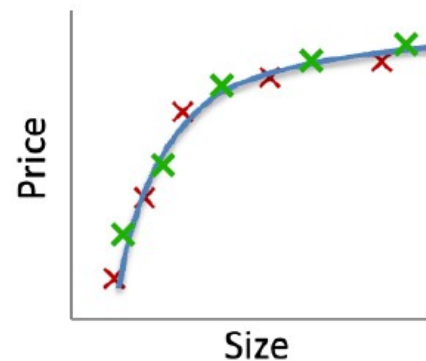
$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$$

High bias
(underfit)

$$d = 1$$

J_{train} is high

J_{cv} is high



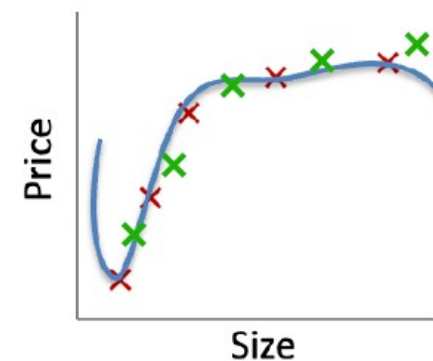
$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + b$$

"Just right"

$$d = 2$$

J_{train} is low

J_{cv} is low



$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + w_3x^3 + w_4x^4 + b$$

High variance
(overfit)

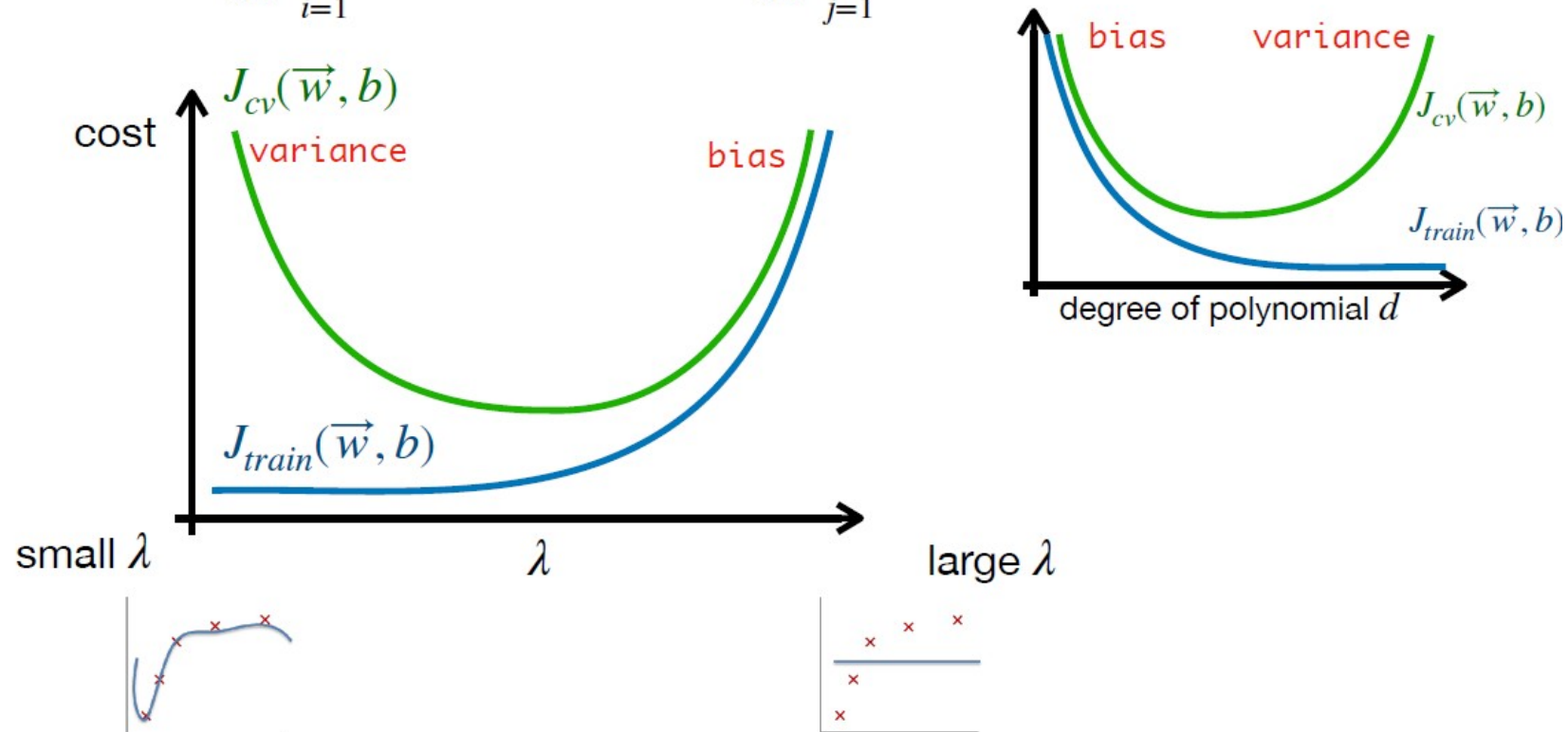
$$d = 4$$

J_{train} is low

J_{cv} is high

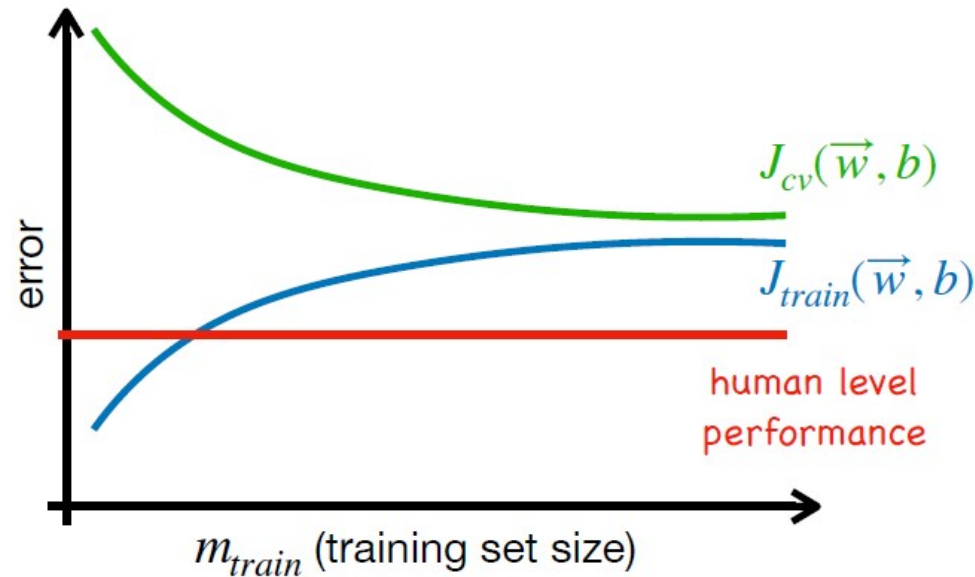
Bias/variance as a function of regularization parameter λ

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

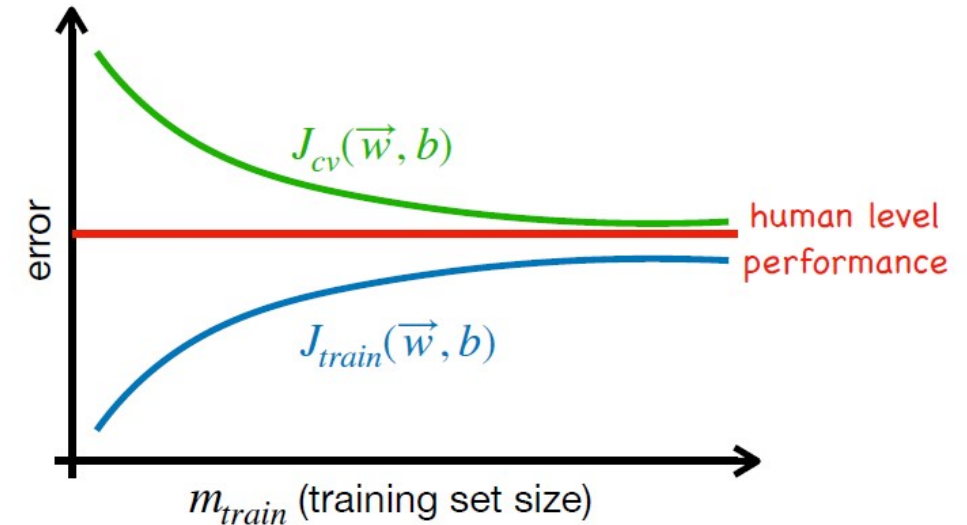


One of the challenges with building machine learning systems is that there's so many things you could try, so many things you could change (hyperparameters)

4.4.1 Using baseline performance

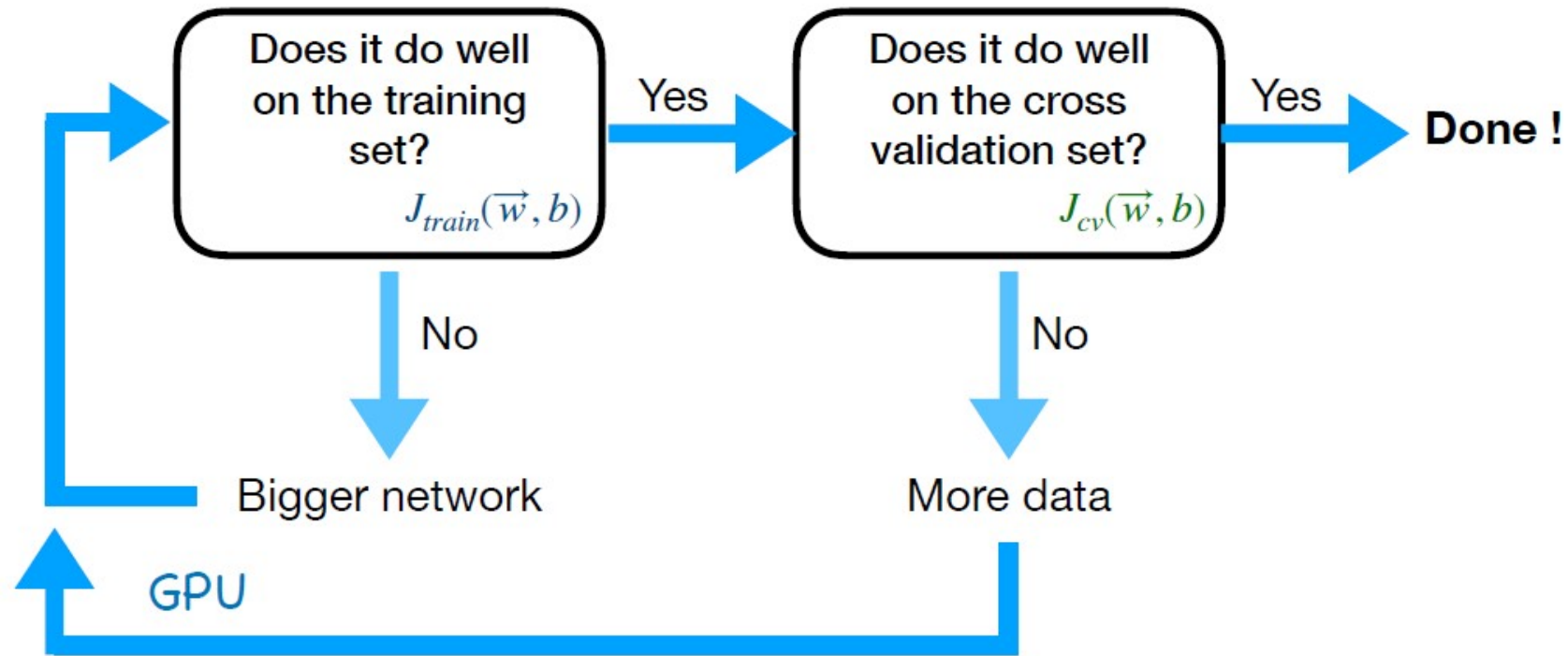


if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much



if a learning algorithm suffers from high variance, getting more training data is likely to help

4.4.2 Neural networks and bias variance



Las grandes redes neuronales son modelos de bajo sesgo. Es importante seleccionar el parámetro de regularización correcto.