MACHINE LEARNING EN FINANZAS

UNIVERSIDAD TORCUATO DI TELLA

LIONEL MODI, CFA

lionel.modi@utdt.edu

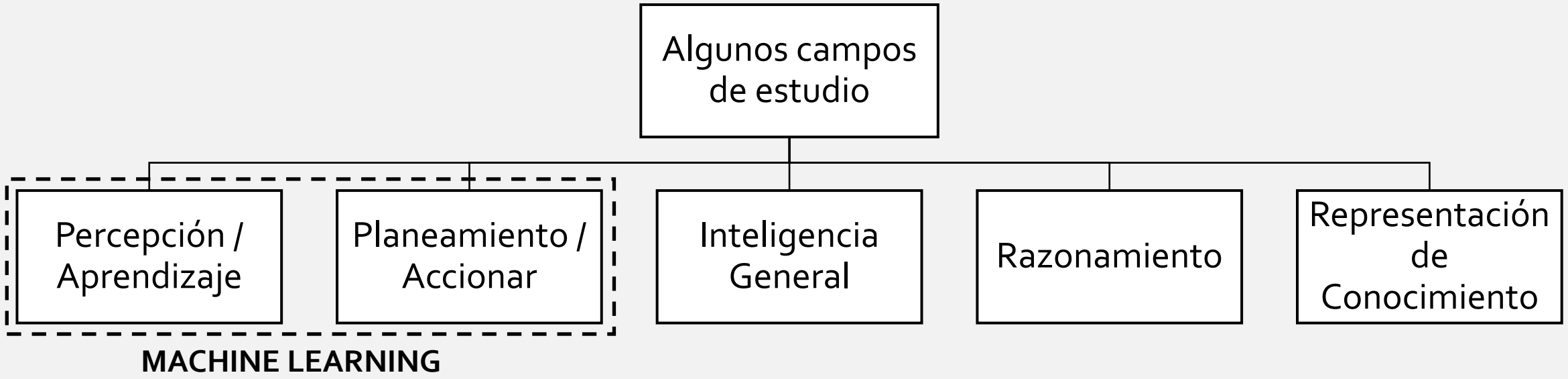
Fundamentos

INTELIGENCIA ARTIFICIAL

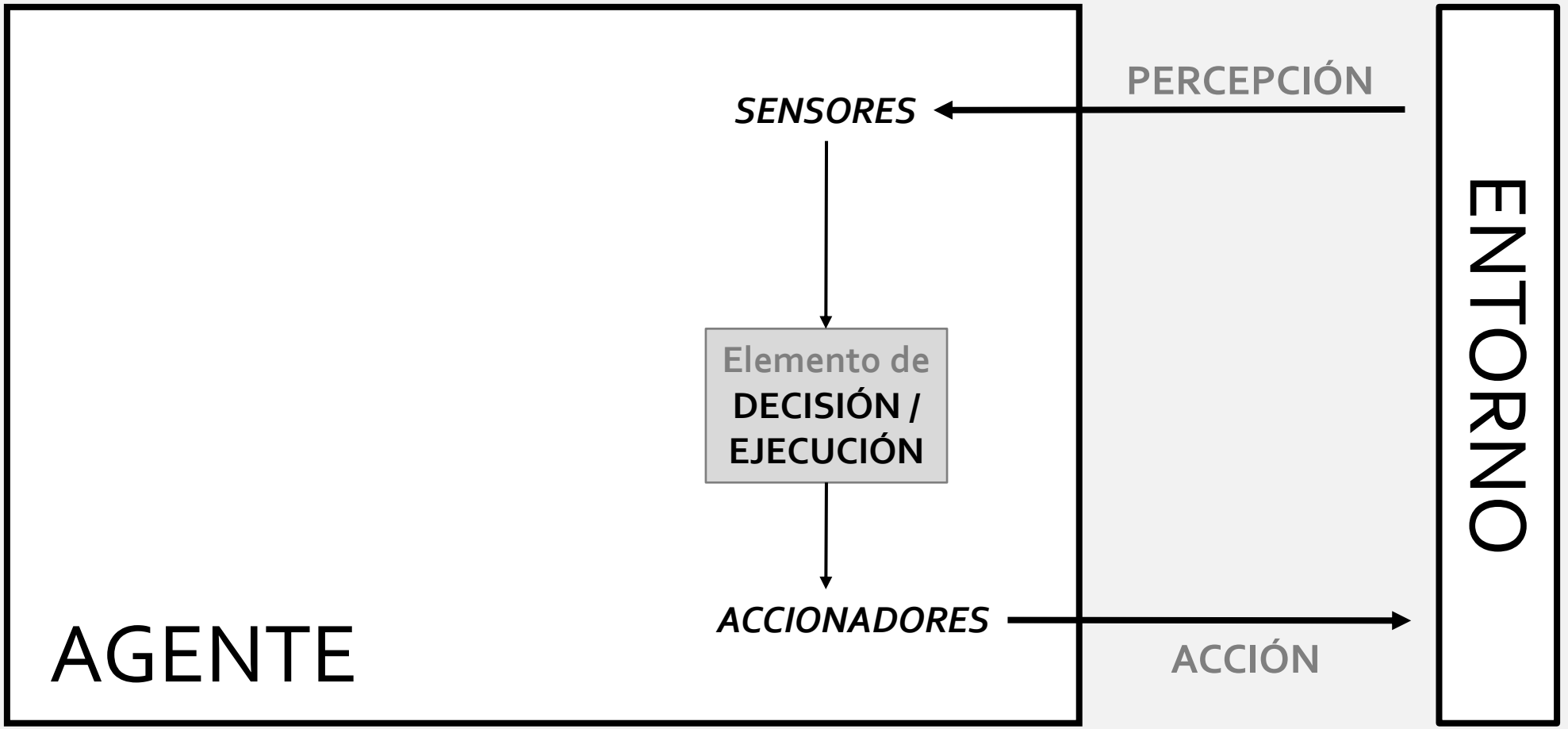
"Estudia el problema de la construcción de agentes que actúen inteligentemente."

"Idealmente, un agente inteligente toma la mejor acción posible ante una situación determinada."

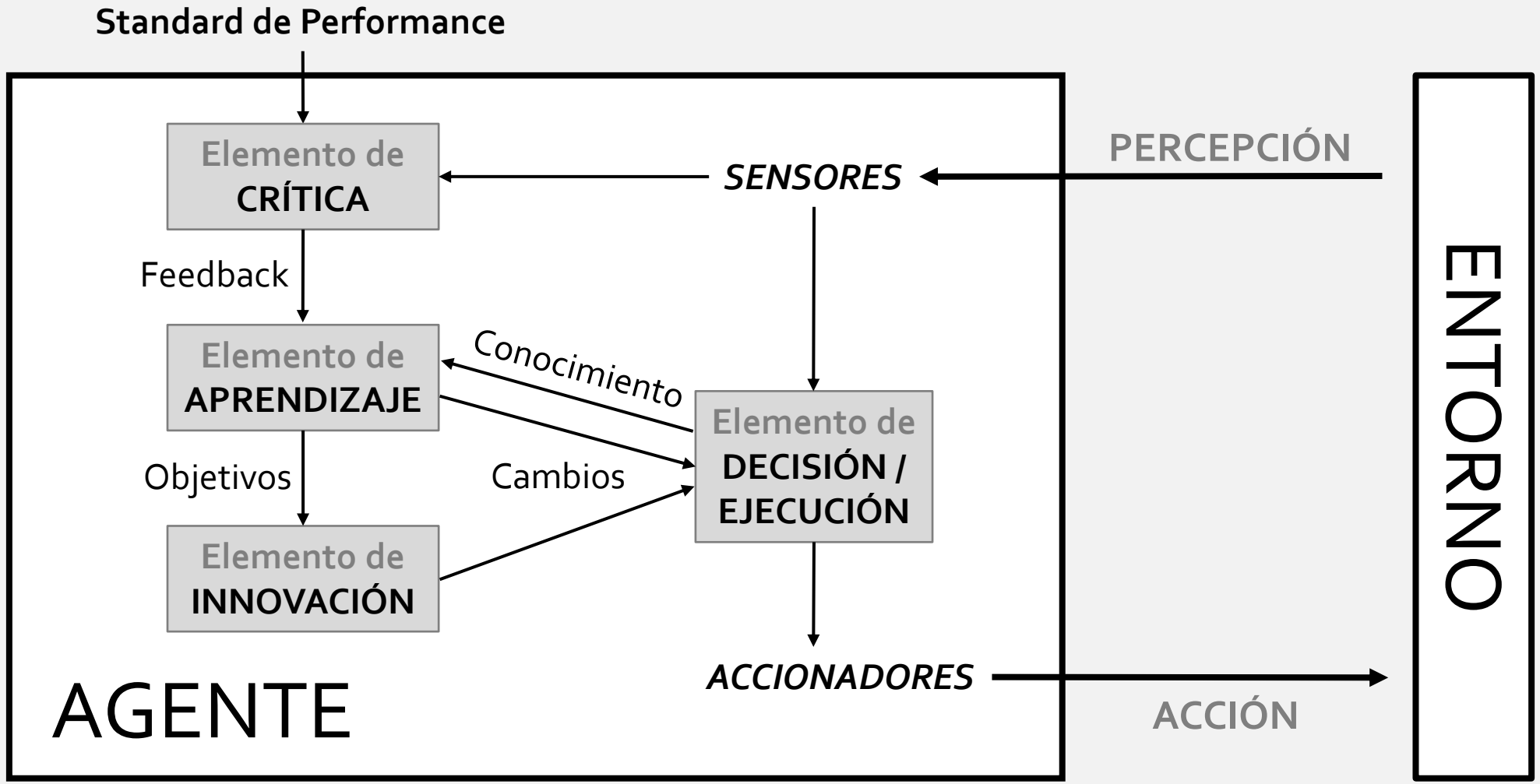
S. Russell & P. Norvig



AGENTE DETERMINÍSTICO



AGENTE QUE APRENDE



QUÉ ES MACHINE LEARNING?

"Es la ciencia (y arte) de programar computadoras para que puedan aprender de los datos."

A. Géron

"Un programa de computadora se dice que aprende de una experiencia **E** con respecto a una tarea **T** y una medida de performance **P**, cuando su performance en **T**, medida por **P**, mejora con la experiencia **E**"

T. Mitchell

TIPOS TAREAS Y MEDIDAS DE PERFORMANCE

TAREAS

con foco en la **PERCEPCIÓN**

La acción es una función preestablecida.

Ejemplos CLASIFICACIÓN y REGRESIÓN

con foco en la **ACCIÓN**

La acción no es fija, sino que existen múltiples alternativas. Incluso puede ser secuencial.

MEDIDAS DE PERFORMANCE

Son específicas de cada tarea y deben ser apropiadas para el fin buscado.

Ejemplos:

Mean Square Error

L1-loss

Error Rate

CONSTRUCCIÓN DE EXPERIENCIA

SUPERVISED LEARNING

Existe guía.
El training set posee pares (**features**; **label**)

UNSUPERVISED LEARNING

No existe guía.
El training set no posee **labels**

REINFORCEMENT LEARNING

Existe guía.
El training set no posee **labels**, pero existe un mecanismo de feedback parcial (**reward**)

PRINCIPALES PROBLEMAS DE ML EN DETALLE

Tareas de PERCEPCIÓN				Tareas de ACCIÓN	
Supervised Learning		**Unsupervised Learning**		**Reinforcement Learning**	
Regresión	Clasificación	Clustering	Representación	Aprendizaje de estrategia	Aprendizaje de objetivos (IRL)
Aprender función de regresión: $f: \mathbb{R}^n \rightarrow \mathbb{R}$ Dados: Pares inputs/ouput $(\boldsymbol{X}_i, Y_i)$	Aprender función de clasificación: $f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$ Dados: Pares inputs/categoría $(\boldsymbol{X}_i, C_i)$	Aprender función de clasificación: $f: \mathbb{R}^n \rightarrow \{1, \dots, k\}$ Dados: Sólo inputs $(\boldsymbol{X}_i)$	Aprender función de representac.: $f: \mathbb{R}^n \rightarrow \mathbb{R}^k$ $k \ll n$ Dados: Sólo inputs $(\boldsymbol{X}_i)$	Aprender función de política: $f: \mathbb{R}^n \rightarrow \mathbb{R}^k$ Dados: Tuplas $(\boldsymbol{X}_i, a_i, \boldsymbol{X}_{i+1}, r_i)$	Aprender función de reward: $f: \mathbb{R}^n \rightarrow \mathbb{R}$ Dados: Tuplas $(\boldsymbol{X}_i, a_i, \boldsymbol{X}_{i+1})$

MACHINE LEARNING EN FINANZAS

El campo de las finanzas suele ofrecer mayores desafíos para el uso de ML.

	Ciencias	Finanzas
Datos IID / Estacionarios	Típicamente **SI**	Típicamente **NO**
Ratio Señal-Ruido	Típicamente **ALTO**	Típicamente **BAJO**
Tareas de Acción	Campo de accionar **reducido**, **baja** incertidumbre	Campo de accionar **muy amplio**, **alta** incertidumbre
Interpretabilidad de Resultados	Típicamente **no muy relevante**	Típicamente **deseada** (trading) o **requerida** (regulación)

Introducción al herramental

TEOREMA DE NO-FREE LUNCH

> "Todos los modelos poseen el **mismo nivel de error** cuando se promedian sus resultados sobre **todas** las posibles distribuciones generadores de datos."
>
> D. Wolpert (1996)

Como consecuencia,

a priori **NO hay garantía** de que un modelo sea siempre el mejor de todos entonces, habría que probar cada uno para ver cuál resulta mejor, lo que se torna empíricamente **imposible**.

Por lo tanto,

1) en la práctica se realizan supuestos **razonables** sobre los datos y,

2) se evalúan **sólo** algunos modelos que resulten adecuados a partir de éstos.

GENERALIZACIÓN

Refiere a la capacidad de un modelo de representar apropiadamente, en términos del valor esperado de su medida de performance, **tanto los datos observados como los NO observados**.

Un resultado teórico clave es la llamada **descomposición BIAS-VARIANCE**:

$$GENERALIZATION\ ERROR = BIAS^2 + VARIANCE + ERROR$$

BIAS *(sesgo)*	Refiere al error producido por usar un modelo que no aproxima tan bien los datos.
VARIANCE *(varianza)*	Refiere a la magnitud del cambio en el modelo estimado a partir de cambios en los datos de estimación.
ERROR	Refiere al error irreducible

COMPARACIÓN DE MODELOS

El **error de generalización** nos permite establecer un criterio de comparación de los modelos a utilizar. Cuanto menor sea este, mejor será el modelo.

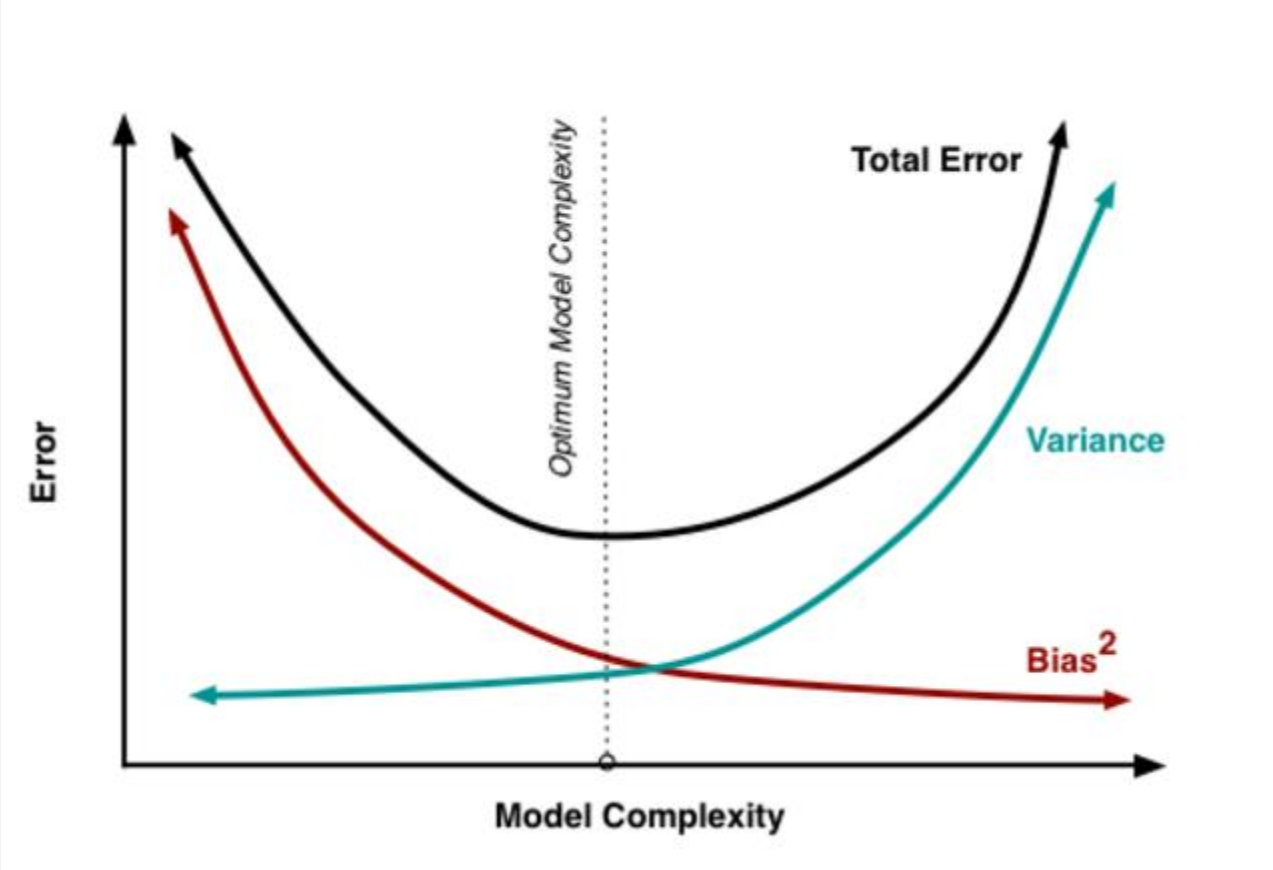
STRONG LEARNER	Alto poder de generalización
WEAK LEARNER	Apenas superior a una predicción aleatoria

Pero el problema que surge al querer optimizar este criterio es el llamado **BIAS-VARIANCE TRADE-OFF.**

Típicamente al **incrementar la complejidad** de un modelo **aumenta su varianza pero se reduce su sesgo**. (OVERFITTING)

Inversamente, **disminuir la complejidad** de un modelo tiende a **reducir su varianza pero a aumentar su sesgo**. (UNDERFITTING)

BIAS-VARIANCE TRADE-OFF



RESUMEN DE MÉTODOS DE MACHINE LEARNING

Tareas de PERCEPCIÓN				Tareas de ACCIÓN	
Supervised Learning		**Unsupervised Learning**		**Reinforcement Learning**	
Regresión	Clasificación	Clustering	Representación	Aprendizaje de estrategia	Aprendizaje de objetivos (IRL)
Linear Regression Penalized Regression Regression Tree SVR Ensemble methods **Neural Networks**	Logistic Regression Naïve Bayes K Nearest Neighbours SVM Classification Tree Ensemble Methods **Neural Networks**	K-Means Hierarchical clustering Gaussian Mixtures Hidden Markov Models **Neural Networks**	Principal Components Analysis Kernel-PCA Independent Components Analysis **Neural Networks**	Model-based RL Model-free RL Batch/Online RL RL with linear models **Neural Networks**	Model-based IRL Model-free IRL Batch/Online IRL MaxEnt IRL **Neural Networks**

Proceso de Aprendizaje

LOS DATOS: DATA SCIENCE

Data Science es la ciencia y arte de trabajar con los datos y extraer features.

> "Los datos importan más que los algoritmos en problemas complejos"
>
> P. Norvik (2009)

Algunos problemas típicos

CANTIDAD INSUFICIENTE	NO REPRESENTATIVOS	MALA CALIDAD	FEATURES INAPROPIADOS
Curse of Dimensionality	Muestra sesgada Muestra desbalanceada	Outliers Features faltantes o incompletas	**Feature Engineering** (selección, extracción, creación y **normalización**)

FEATURE ENGINEERING

“How, When, and Why Should You Normalize / Standardize / Rescale Your Data?”

https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff

“Normalization and Standardization in 2 Minutes”

https://towardsdatascience.com/normalization-and-standardization-in-2-minutes-e0609a01e76

TRAINING Y TESTING

Para el aprendizaje, se utilizan dos set de datos

TRAINING SET

Utilizado para entrenar/estimar el modelo minimizando el **Training Error**.

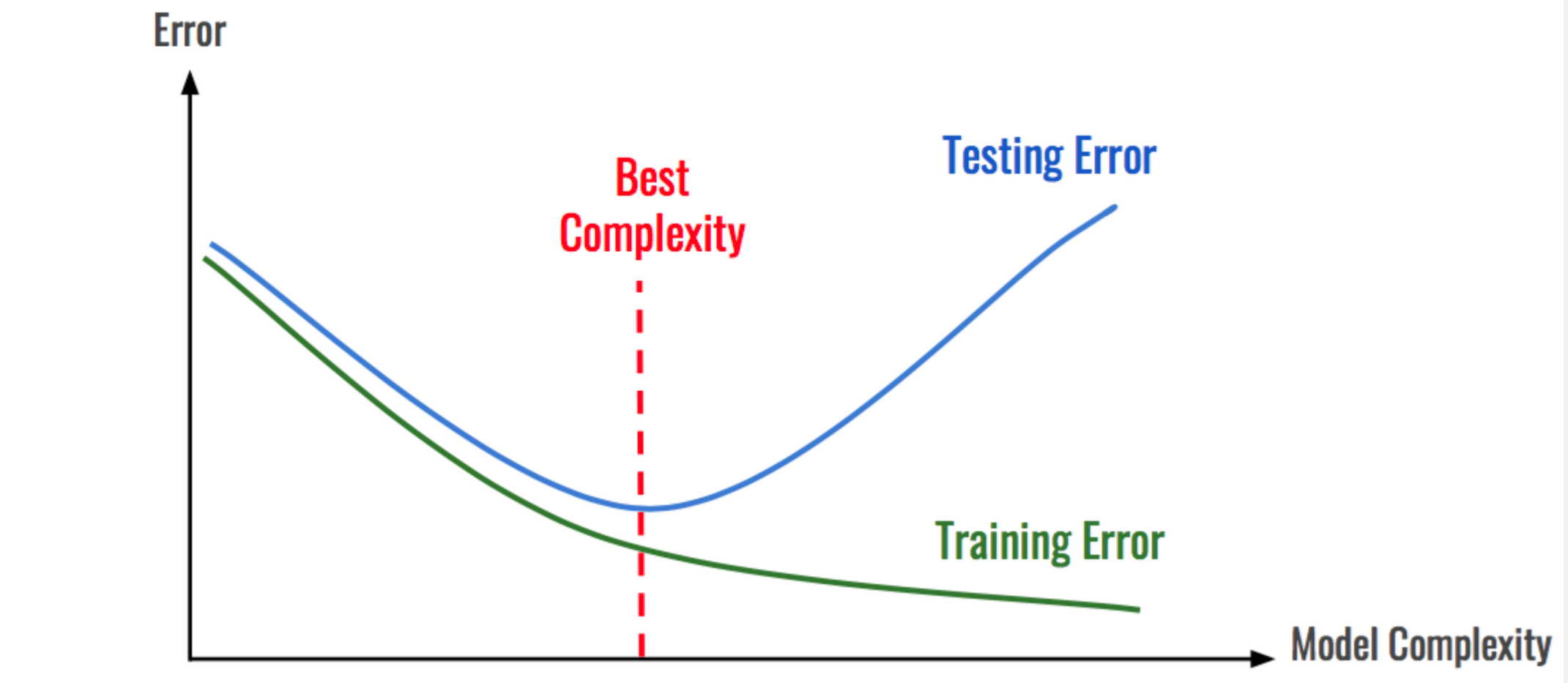
TESTING SET

Utilizado para estimar la capacidad del algoritmo de generalizar **out-of-sample**.
Permite estimar el **Generalization Error** a partir del **Test Error**.

El proceso de trabajo en ML se puede resumir entonces como:

1. Entrenar un modelo con el training set (**Batch vs. Online**).
2. Estimar el error de generalización usando el testing set.
3. Comparar performance con otros modelos.

REGULANDO LA COMPLEJIDAD DE UN MODELO



REGULARIZACIÓN E HIPERPARÁMETROS

Se denomina **regularización** al proceso de restringir un modelo para que resulte más simple y con menor riesgo de overfitting.

El grado de regularización a aplicar durante el aprendizaje es controlado mediante **hiperparámetros**.

Los hiperparámetros son parámetros **del algoritmo de aprendizaje**, **no del modelo** a aprender.

A diferencia de los parámetros standard del modelo, éstos se **fijan** previo al training, y se mantienen **constantes** durante todo dicho proceso.

De forma de calibrarlos, se repiten procesos de training-testing para distintos valores. Luego se comparan resultados.

PROBLEMA DE OPTIMIZACIÓN GENERAL

En resumen, el **algoritmo de aprendizaje** de nuestro modelo procurará resolver, posiblemente en etapas, el siguiente problema **conceptual**

$$\min_{\beta,\lambda} Loss(\beta) + Regularization(\beta, \lambda)$$

donde

$\boldsymbol{\beta}$ son los parámetros del modelo, y $\boldsymbol{\lambda}$ son los hiperparámetros

$\boldsymbol{Loss(\beta)}$

Representa la penalización por el error de generalización (controla el **sesgo**).

$\boldsymbol{Regularization(\beta, \lambda)}$

Representa la penalización por complejidad del modelo (controla la **varianza**).

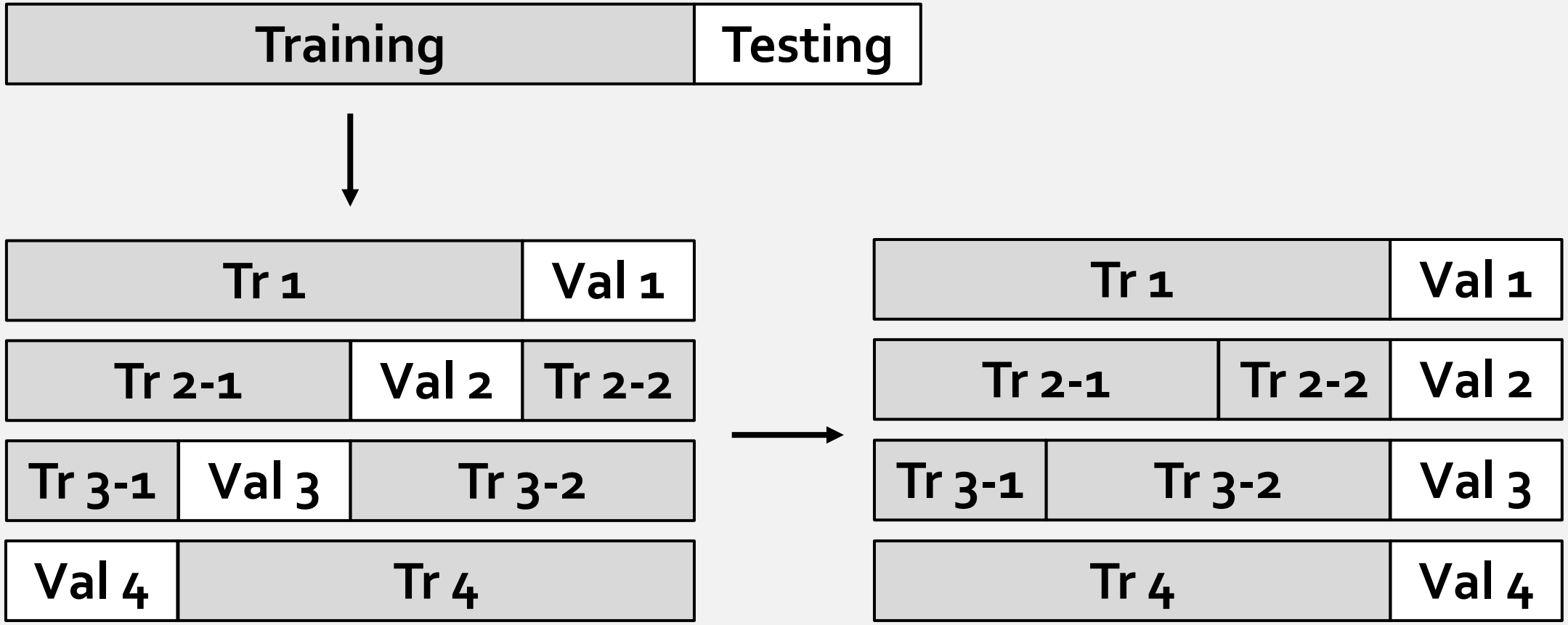
VALIDACIÓN POR RESAMPLING

Al evaluar múltiples modelos utilizando el Testing Set se corre riesgo de hacer **overfitting** de este último. Para evitar esto, se suele extraer del Training Set, otro set de datos para **validación**, que actuará como Testing Set **simulado**.

> **Cross-Validation** es una técnica de resampling que minimiza el costo del Validation Set a partir de la separación del **Training Set** en subconjuntos complementarios, parar luego entrenar-validar los modelos sobre diferentes combinaciones de éstos.

Seleccionados el modelo e hiperparámetros del algoritmo de aprendizaje, finalmente se entrena sobre el **Training Set** completo y se valida contra el **Testing Set**.

CROSS-VALIDATION: GRÁFICAMENTE



CROSS-VALIDATION: COMENTARIOS

Existen múltiples variantes (ej. LOO, K-Fold, etc.)

Idealmente se debe preservar en las submuestras las características estructurales de la muestra original.

Utilizar **estratificación** si la muestra original no es balanceada

Si los datos no son **i.i.d.** (problema usual en finanzas) se debe corregir por dicho problema (ej. ver López de Prado)

ENSEMBLE LEARNING

Es una técnica de ML que implica el uso de un grupo de modelos (ensemble) en forma **colaborativa**.

Está empíricamente demostrado que los **Ensemble Methods** suelen poseer **mejor** poder predictivo que modelos individuales **cuando** los métodos pertenecientes al ensemble son **disimilares** entre sí.

Algunos formatos

Voting	**Bagging**	**Boosting**	**Stacking**
Modelos distintos en paralelo.	Modelos iguales en paralelo pero con diferentes submuestras	Modelos iguales en forma secuencial sobre error previo	Modelos distintos en paralelo con modelo *blender* para weighting.

Machine Learning en Python

ENTORNO

Vamos a estar utilizando para el curso

VISUAL STUDIO CODE

https://code.visualstudio.com/Download

Extensiones:

- **Python** (Microsoft) – requerida
- **Excel Viewer** (GrapeCity) – recomendada

ANACONDA / MINICONDA (Python 3.x)

https://www.anaconda.com/distribution/

https://docs.conda.io/en/latest/miniconda.html

PRINCIPALES PAQUETES DE PYTHON

Los paquetes que utilizando en el curso son del canal `conda-forge`

	conda package	Namespace	Contenido
scikit-learn	`scikit-learn`	`sklearn`	Métodos de ML tradicionales y framework general para trabajar en ML en Python
TensorFlow	`tensorflow` (`tensorflow-gpu`)	`tensorflow`	Engine de cálculo numérico con especial aplicación en Redes Neuronales. **Incluye el API Keras.**
XGBoost	`xgboost`	`xgboost`	Librería específica de Boosting de alta performance
optuna	`optuna`	`optuna`	Librería para calibración de hiperparámetros

CARACTERÍSTICAS DEL API DE SCIKIT-LEARN

Clases consistentes	**Estimator**	Puede estimar parámetros a través de `fit()`
	Transformer	Estimadores con transformación datos a través de `transform()`
	Predictor	Pueden realizar predicciones a través de `predict()`
Fácil acceso	**Hiperparámetros**	Accesibles vía member variables
	Parámetros	Accesibles vía member variables con underscore final.
Uso de clases standard	**Datasets**	`numpy` arrays o `scipy` sparse matrices
	Hiperparámetros	strings o números
Composición	Clases existentes altamente reutilizables.	
Default values apropiados	Los parámetros base suelen estar pre-configurados para estimación rápida.	

En nuestro curso procuraremos desarrollar nuestro código en **sintonía** con este API para maximizar reutilización y generalidad.

PROCESO DE APRENDIZAJE EN SCIKIT-LEARN

	Tipo	En namespace `sklearn`
PREPROCESAMIENTO	Módulo	`preprocessing`
	Módulo	`feature_extraction`
MUESTREO	Función	`model_selection.train_test_split`
VALIDACIÓN DE MODELOS	Función	`model_selection.cross_validate`
	Función	`model_selection.cross_val_score`
	Función	`model_selection.learning_curve`
CALIBRACIÓN DE HIPERPARÁMETEROS	Clase	`model_selection.GridSearchCV`
	Clase	`model_selection.RandomizedSearchCV`

Artificial Neural Networks

INTRODUCCIÓN

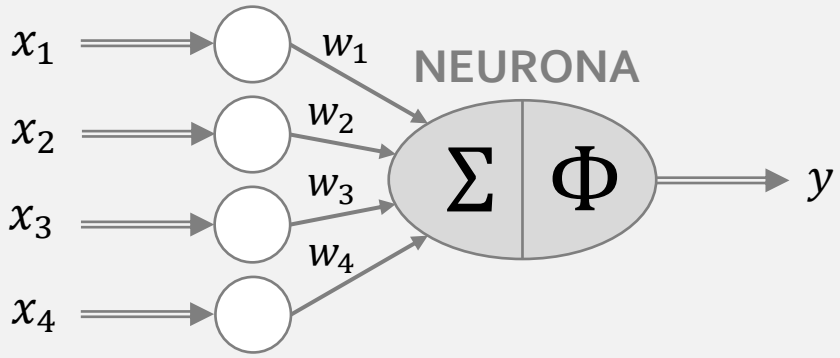
Las redes neuronales artificiales (NN) son una técnica de ML que busca simular el mecanismo de aprendizaje en organismos biológicos.

Combina **unidades de cómputo** a las que llamamos neuronas.

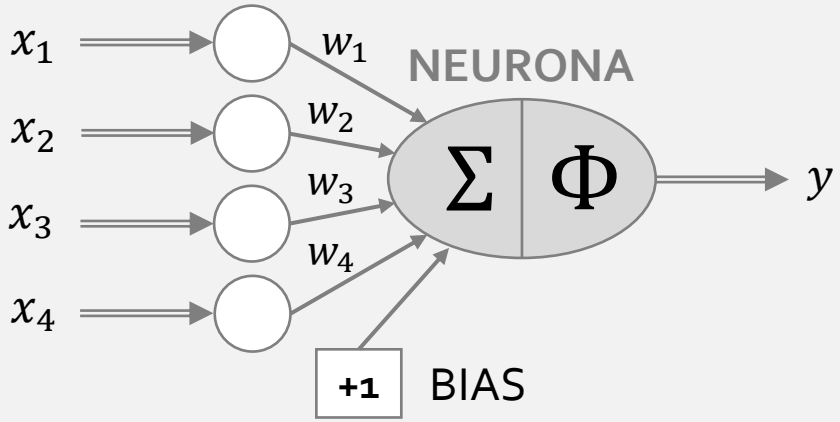
> Las NN logran a partir de la combinación de capas y neuronas **aproximar casi cualquier función** matemática (y modelo de ML) , esto las hace extremadamente flexibles y de gran poder.

Desafortunadamente su generalidad suele tener fuerte impacto en la **performance** de su entrenamiento, por lo que hay que considerar con detenimiento su diseño.

ARQUITECTURA BÁSICA



Sin BIAS

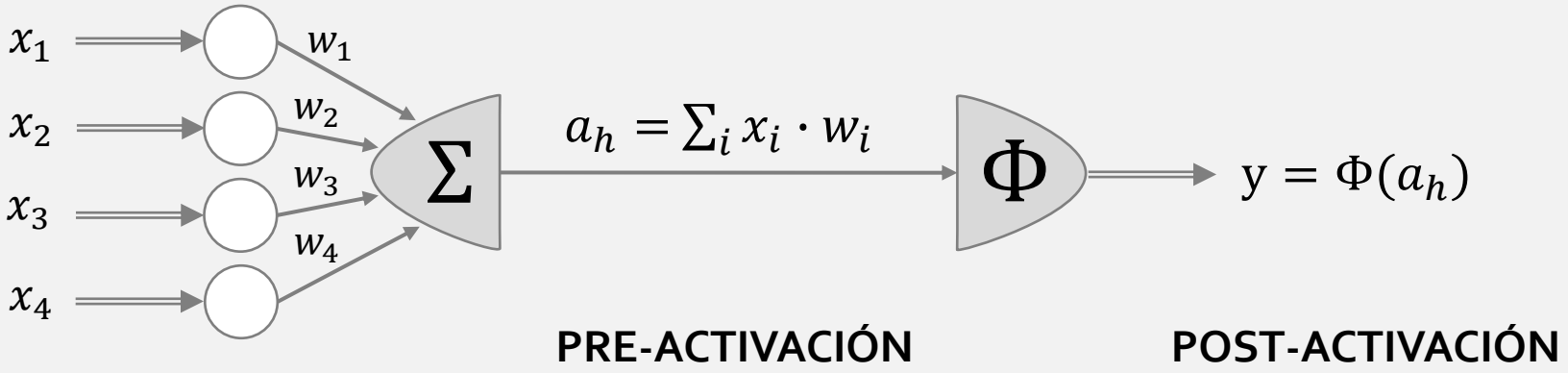


Con BIAS

Posee una capa (layer) de entrada y una capa de cómputo compuesta por sólo una **neurona** (que coincide en este caso con la capa de salida).

Al tipificar la arquitectura de una NN, **sólo cuentan las capas de cómputo**, por lo que este modelo se considera de una única capa. Una arquitectura de 1 o 2 capas se denomina **shallow**.

PRE Y POST ACTIVACIÓN



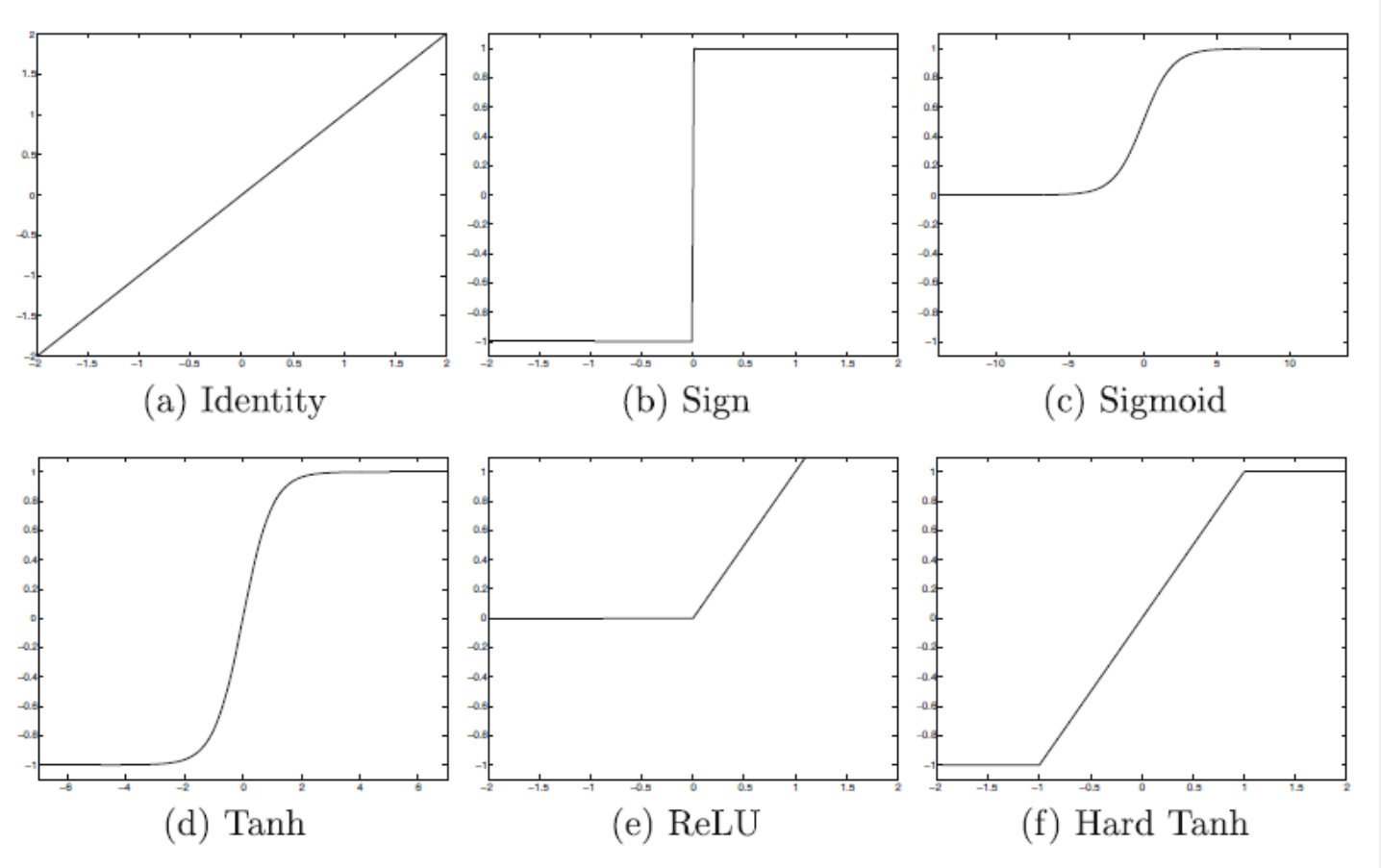
Esta división conceptual expone la naturaleza compuesta de las operaciones matemáticas implícitas en una NN.

$$y = \Phi(\sum_i x_i \cdot w_i) = \Phi\big(a_h(x)\big)$$

Donde Φ se conoce como **función de activación**.

FUNCIONES DE ACTIVACIÓN

La correcta elección de la función de activación nos permite aproximar comportamientos **no lineales** con nuestra NN.



$(a)\ \Phi(x) = x$

$(b)\ \Phi(x) = sign(x)$

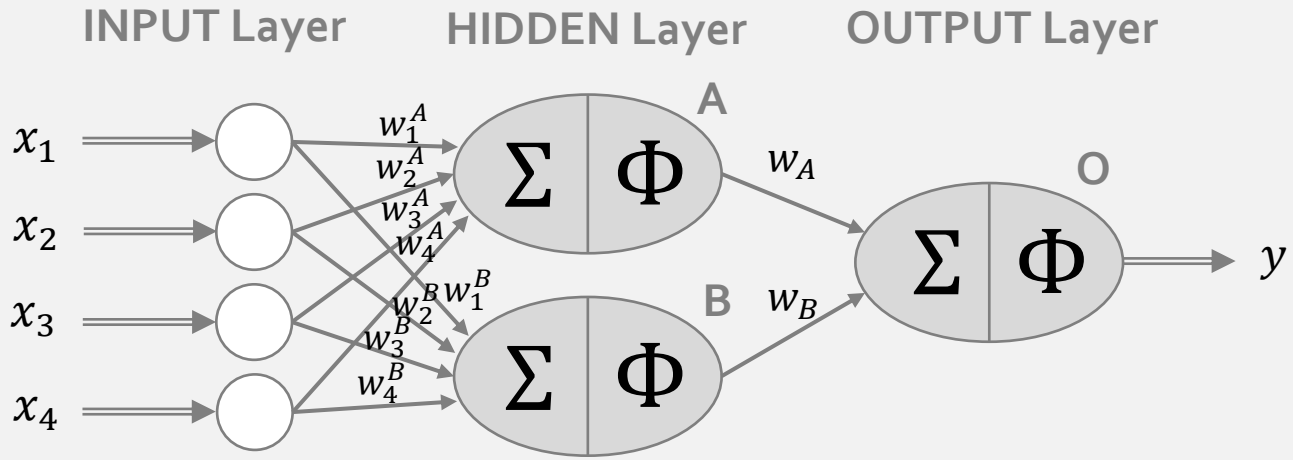
$(c)\ \Phi(x) = \dfrac{1}{1 + e^{-x}}$

$(d)\ \Phi(x) = \dfrac{e^{2x} - 1}{e^{2x} + 1}$

$(e)\ \Phi(x) = \max\{x, 0\}$

$(f)\ \Phi(x) = \max\{\min[x, 1], -1\}$

ARQUITECTURA MULTI-CAPA



$$y = \Phi_o\left(w_A \cdot \Phi_A\left(\sum_i x_i \cdot w_i^A\right) + w_B \cdot \Phi_B\left(\sum_i x_i \cdot w_i^B\right)\right) = \Phi_o(w_A \cdot \Phi_A(A_h(x)) + w_B \cdot \Phi_B(B_h(x))) = \Phi_o(O_h(x))$$

Esta arquitectura de capas una después de otra (stacked), donde no existe recurrencia, se denomina **feed-forward**.

Una capa que conecta todas sus neuronas con todos los inputs se denomina densa (**Dense**).

La arquitectura es de **DEEP LEARNING** si la hidden layer tiene 2 o más capas.

ENTRENAMIENTO Y BACKPROPAGATION

Finalmente para entrenar la NN, como cualquier otro modelo de ML, necesitamos especificar una **Loss Function** apropiada.

Una vez especificada ésta, se utiliza un algoritmo de aprendizaje particular basado en programación dinámica que se denomina **BACKPROPAGATION.**

> Es clave para que al algoritmo de aprendizaje cambie los weigths en forma correcta obtener el **gradiente** de la NN el cual se calcula aplicando la **regla de la cadena** de derivadas a través de cada función de activación anidada.

Cuanto más compleja y profunda la red, más costoso y difícil será el entrenamiento con backpropagation (ej. **vanishing/exploding gradient**)

Algortimos de Aprendizaje y Optimización Numérica

ALGORITMOS DE APRENDIZAJE

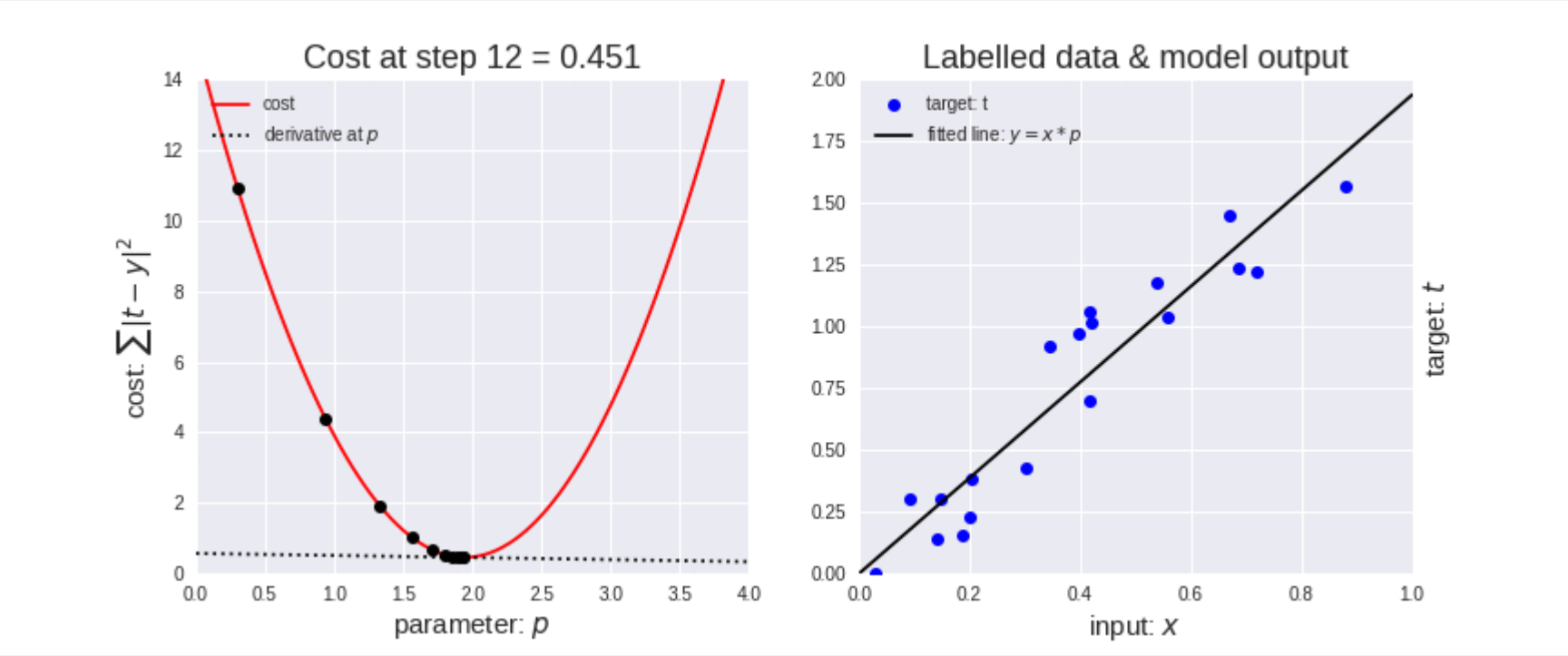
Varios de los modelos de ML poseen algoritmos de aprendizaje **específicos** diseñados para maximizar la eficiencia. En algunos casos incluso existen soluciones **algebraicas** (ej. ecuaciones normales).

Otros, particularmente las NN, utilizan algoritmos de propósito general.

GRADIENT DESCENT	Minimizan una función modificando los valores de los paráme-tros en el sentido opuesto al gradiente de la función.
EVOLUTIONARY	Minimizan una función utilizando mecanismos inspirados en la evolución biológica (ej. mutación, selección, combinación, etc.). Poseen especial aplicación en **Deep Learning**

(ver https://towardsdatascience.com/gradient-descent-vs-neuroevolution-f907dace010f)

GRADIENT DESCENT EN ACCIÓN



VARIANTES DE GD Y MÉTODOS DE OPTIMIZACIÓN

Existen tres variantes básicas que se diferencian en cuántos datos utilizan para realizar la optimización.

Batch GD **Stochastic GD (SGD)** **Mini-batch GD**

El criterio de selección entre uno oscila entre precisión vs. velocidad.

Asimismo existen numerosos métodos de optimización numérica utilizados para el algoritmo GD.

Momentum **Nesterov** **Adagrad** **RMSprop** **Adam**

(ver http://ruder.io/optimizing-gradient-descent/index.html)

NN en Python
TensorFlow

INTRODUCCIÓN

TF basa su modelo de cálculo en el conceptos de **Graphs**.

Los Graphs combinan **Tensors** (vectores) con operaciones matemáticas para construir una **expresión** que luego puede calcularse en forma **lazy** o **eager**.

TF **no** sólo es una librería para NN, puede ser utilizada para matemática en general.

Su diseño permite distribuir el cálculo entre **CPUs**, **GPUs** o **clusters**.

Posee **APIs** para utilizarla desde Python, pero TF ejecuta código nativo (compilado) desarrollado en C++, por lo que evita los cuellos de botella en performance de Python.

Utilizando los APIs más **avanzados** se puede modelar casi cualquier NN.

API KERAS: MODELANDO NN

El API **Keras** permite modelar NN en TF mediante 3 sencillos pasos

PASO 1

CONSTRUCCIÓN DEL MODELO

Se pueden utilizar dos formatos de sub-APIs: **Sequential** (más básica) o **Functional** (más flexible y poderosa)

PASO 2

COMPILACIÓN DEL MODELO

Se establecen la **Loss Function** y el **algoritmo de optimización** creándose el Graph con la NN.

PASO 3

ENTRENAMIENTO Y EVALUACIÓN

Se realiza el entrenamiento con el método `fit` y la evaluación con `evaluate`

COMENTARIOS ADICIONALES

Para monitorear el proceso de entrenamiento de una NN (particularmente en Deep Learning) y/o visualizar su arquitectura TF provee una utilidad gráfica muy útil llamada **TensorBoard**.

El módulo `tensorflow.keras.wrappers.scikit_learn` posee clases que permiten integrar un modelo de TF en scikit-learn.

El problema de Regresión

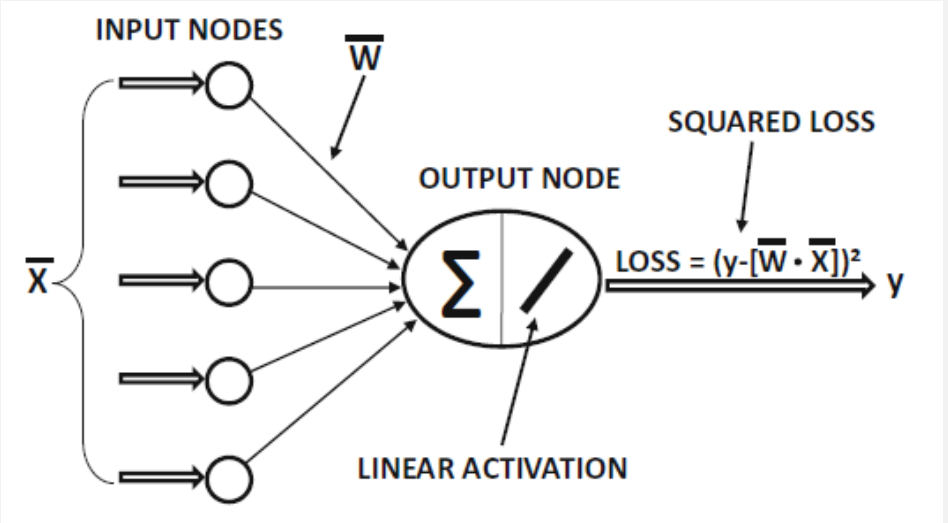
LINEAR REGRESSION

Regresión lineal por OLS clásica.

Modelo	$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j \cdot x_j$
Loss Function	$\sum_{i=1}^{n} (y_i - \hat{y})^2 = RSS$

Clases relacionadas en `sklearn`
`linear_model.LinearRegression`
`linear_model.SGDRegressor(loss='squared_loss', penalty='none')`

LINEAR REGRESSION: NN



Clases relacionadas en `tensorflow`
`keras.layers.Dense(activation='linear', use_bias=True)`
`keras.losses.MSE`
`keras.optimizers.SGD`

PENALIZED REGRESSIONS

Son regresiones lineales donde se aplica una **penalización** (regularización) a los parámetros para aumentar el poder de generalización.

> La penalización se implementa a partir de acercar (**shrink**) a cero los parámetros que no poseen poder predictivo.

Cabe notar que el shrinking vuelve a estas regresiones **sensibles** a la escala.

La particularidad que diferencia a un modelo de otro es la forma en cómo se mide la distancia hacia el cero. Las dos distancias básicas son

Nombre Coloquial	Nombre Técnico	Fórmula
Manhattan	$\ell_1\ norm$	$\|\beta\|_1 = \sum_{j=1}^{p} \|\beta_j\|$
Euclideana	$\ell_2\ norm$	$\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$

LASSO Y RIDGE REGRESSIONS

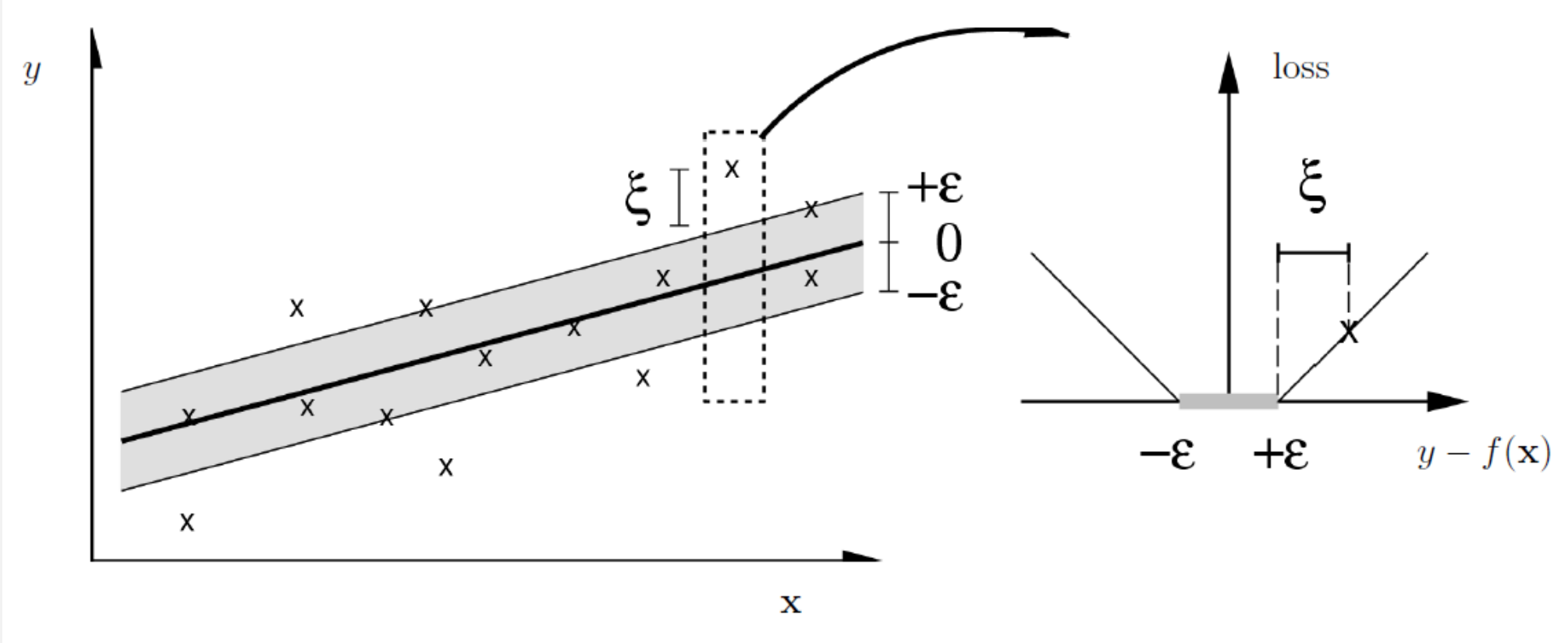
Modelo	$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j \cdot x_j$

	LASSO	RIDGE
Loss Function	$RSS + \lambda \cdot \sum_{j=1}^{p} \lvert\beta_j\rvert$	$RSS + \lambda \cdot \sum_{j=1}^{p} \beta_j^2$
Razgo particular	Genera modelos con subset de los parámetros (**sparse**). (Feature Selection)	Genera modelos con **todos** los parámetros aún siendo de valores cercanos a cero.

Clases relacionadas en `sklearn.linear_model`	`Lasso`
	`LassoCV`
	`SGDRegressor(`
	`  loss='squared_loss',`
	`  penalty='l1')`
		`Ridge`
		`RidgeCV`
		`SGDRegressor(`
		`  loss='squared_loss',`
		`  penalty='l2')`

SUPPORT VECTOR MACHINE REGRESSION

SVR busca aproximar una función que minimice una Loss Function que contemple cierto umbral (***ϵ*-tube**) en donde no se penalice el error.



SUPPORT VECTOR MACHINE REGRESSION: MODELO

Modelo	$\hat{y} = \beta_0 + \sum_{j=1}^{p} \sum_{m=1}^{M} \beta_{j,m} \cdot h_m(x_j)$
Loss Function	$\mathrm{C} \cdot \max(\lvert y - \hat{y} \rvert - \epsilon, 0)$ (+ shrinking)

donde h es una **basis function** que transforma los regresores para permitir aproximaciones arbitrarias.

Se demuestra que para resolver este problema **no** hace falta especificar estas h, ya que existe una representación dual del problema utilizando **kernels**.

Los kernels son funciones que de miden **similaridad** entre observaciones y son **sensibles** a las escala.

SUPPORT VECTOR MACHINE REGRESSION: KERNELS

Según el kernel utilizado podemos adaptar la forma de nuestro modelo de regression, logrando por ejemplo ajustar comportamientos **no lineales**.

Cada kernel poseen sus propios **hiperparámetros**.

Linear	$K(\boldsymbol{a}, \boldsymbol{b}) = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$
Polynomial	$K(\boldsymbol{a}, \boldsymbol{b}) = (\gamma \langle \boldsymbol{a}, \boldsymbol{b} \rangle + \alpha)^d$
Gaussian RBF	$K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\gamma \|\boldsymbol{a} - \boldsymbol{b}\|^2)$
Sigmoid	$K(\boldsymbol{a}, \boldsymbol{b}) = \tanh(\gamma \langle \boldsymbol{a}, \boldsymbol{b} \rangle + \alpha)$

SUPPORT VECTOR MACHINE REGRESSION: SCIKIT-LEARN

Clases relacionadas en `sklearn`
`svm.LinearSVR`
`svm.SVR`

Para calibrar los hiperparámetros utilizar `optuna` o:

`sklearn.model_selection.GridSearchCV`

`sklearn.model_selection.RandomizedSearchCV`

REGRESSION TREE

Representan información a partir de la subdivisión en regiones del dominio de los predictores generando **segmentos** que luego permiten la predicción.

Poseen una fácil interpretación aunque su poder predictivo no es grande por lo que suelen ser utilizados en **ensembles**.

Algoritmo General

1) Se divide el espacio de los predictores en J regiones no solapadas $R_1, R_2, \ldots, R_J$ de forma **greedy** (la elección de las divisiones responde a lo que más disminuya el RSS **para ese nivel**).
2) La predicción para un caso que cae en la region R_j es igual al promedio de los valores de y para las observaciones de R_j

REGRESSION TREE: MODELO

Modelo	$\hat{y} = \sum_{j=1}^{J} ave(y_i \mid x_i \in R_j) \cdot I(x \in R_j)$
Loss Function	$\sum_{m=1}^{\lvert T \rvert} \sum_{i:x_i \in R_m} (y_i - \hat{y})^2 + \alpha \lvert T \rvert$

Donde $|T|$ representa la cantidad de ramas del árbol, y el ultimo término de la Loss Function implementa **cost complexity pruning**.

Pruning es el proceso de **recortar** el árbol para que generalice mejor.

Otras implementaciones de pruning: **a)** restricción a la cantidad minima de hojas (datos) que cada rama contenga, o **b)** restricción a la profundidad del árbol.

REGRESSION TREE: COMENTARIOS

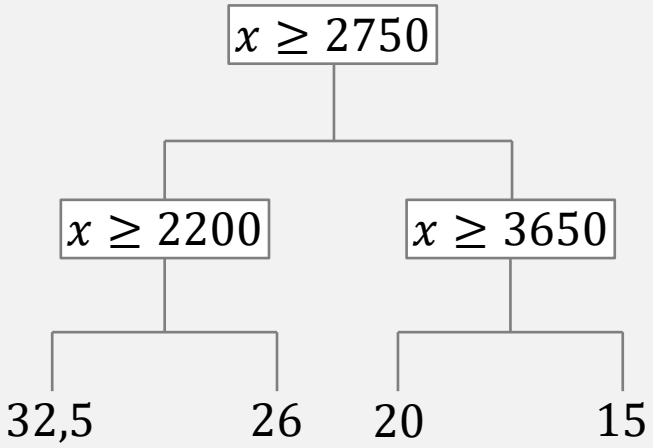
NO son sensibles a la escala

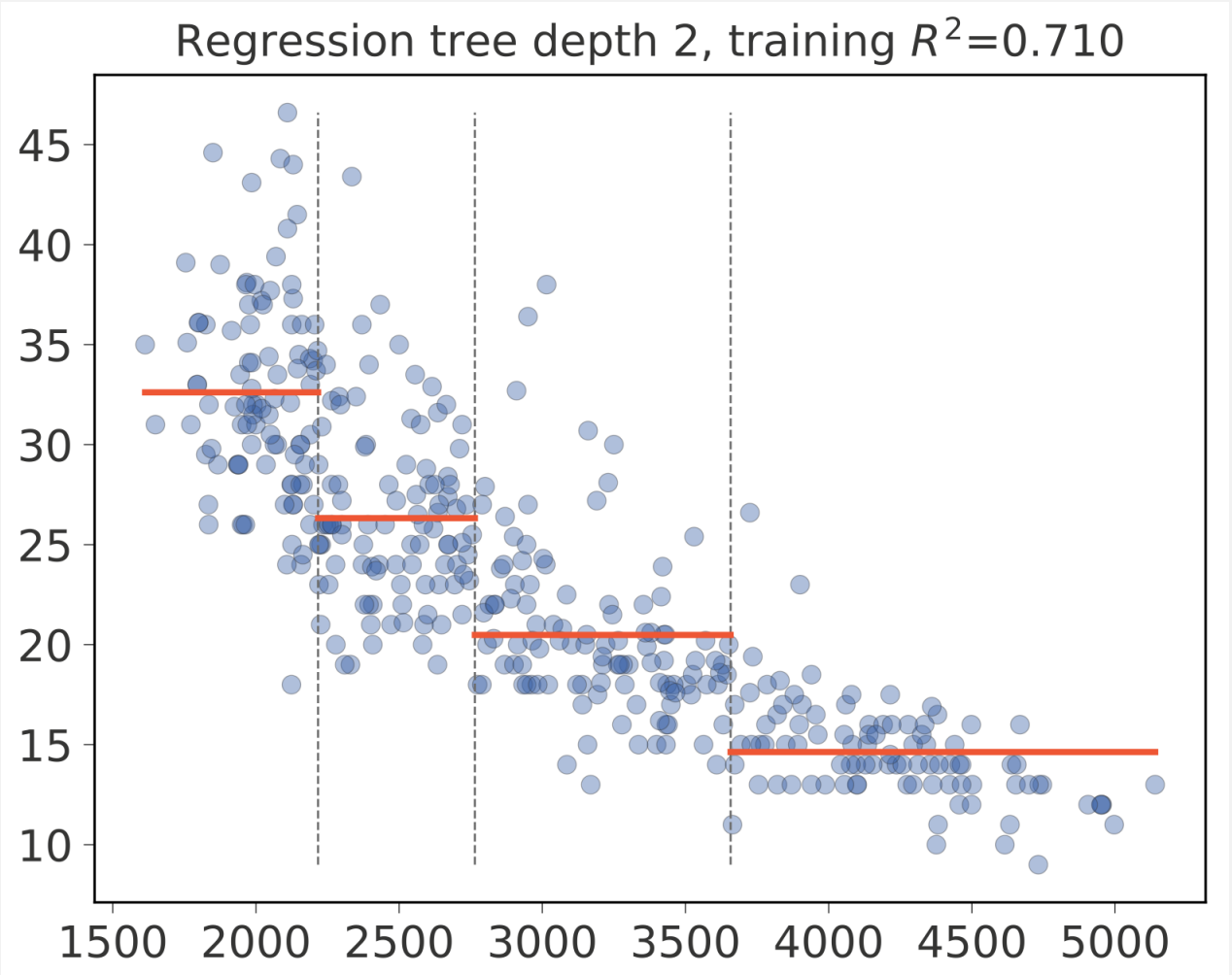
Son **inestables** ante cambios en el dataset (ej. eliminación)

Funcionan mejor cuando los datos son lo más **ortogonales** posibles.

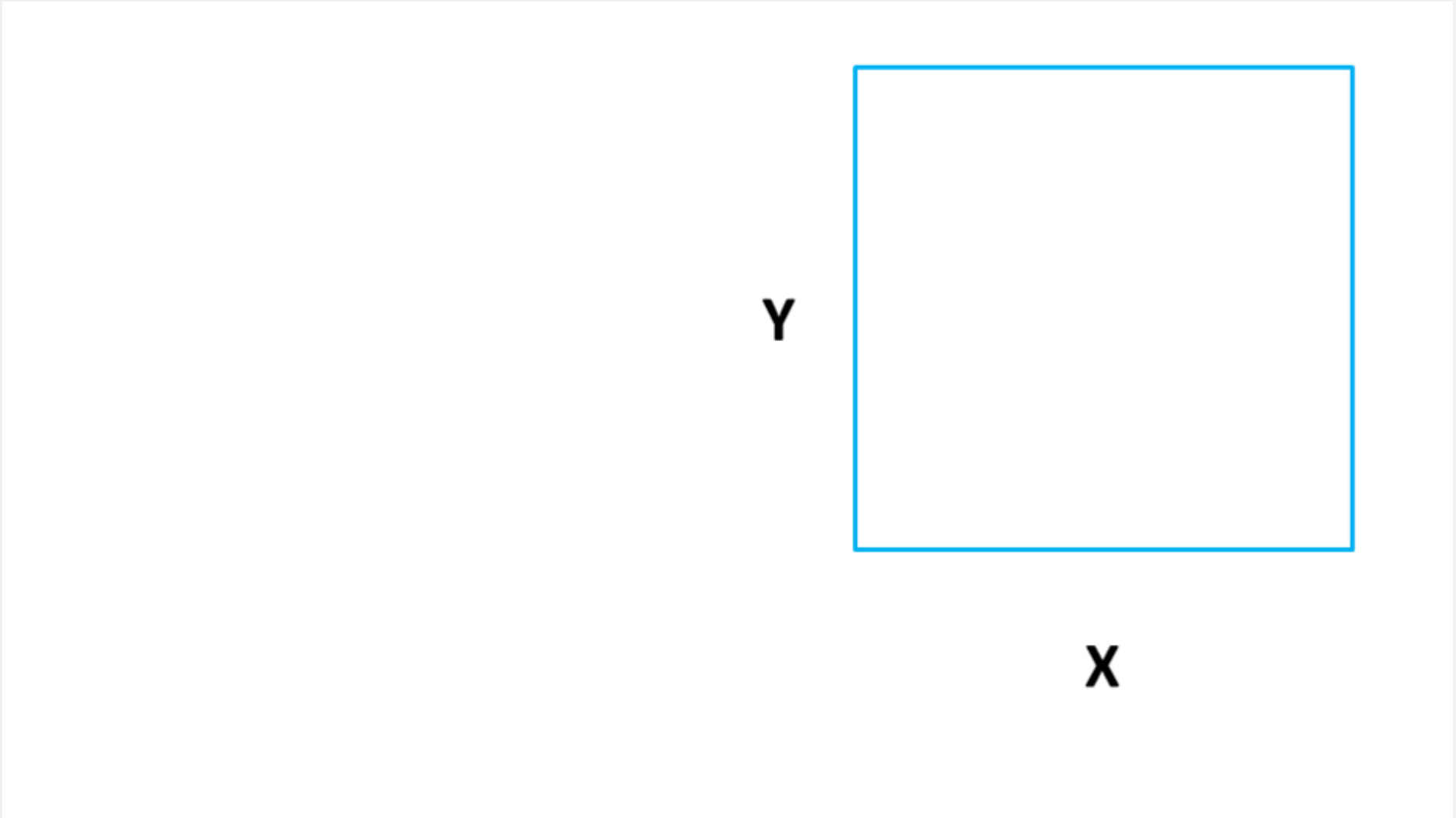
El esquema greedy de construcción genera una jerarquía desde mayor hacia menor en importancia de los regresores que hace a los árboles apropiados para **feature selection**.

REGRESSION TREE EN ACCIÓN PARA 1 PREDICTOR





REGRESSION TREE EN ACCIÓN PARA 2 PREDICTORES



REGRESSION TREE EN SCIKIT-LEARN

Clases relacionadas en `sklearn`
`tree.DecisionTreeRegressor`

Para calibrar los hiperparámetros utilizer `optuna` o:

`sklearn.model_selection.GridSearchCV`

`sklearn.model_selection.RandomizedSearchCV`

El problema de Clasificación

CONCEPTO GENERAL

A diferencia del problema de regresión, en clasificación estaremos buscando predecir **pertenencia** a clases, es decir un campo discreto.

La clasificación puede ser binaria (**binomial**) o en múltiples clasificaciones (**multinomial**).

El proceso interno de los diversos modelos de alguna manera consiste en

- Computar un **decision score** de pertenencia a una clase
- Se prefija un **threshold** para determinar la pertenencia
- Se elabora la predicción y se revisa la **performance**

MEDIDAS DE PERFORMANCE COMUNES

Error Rate	$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$

Accuracy	$1 - Error\ rate$

Confusion Matrix

		Predicción: Negative	Predicción: Positive
Real: Negative		TN	FP
Real: Positive		FN	TP

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

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

$\boldsymbol{F_1}$	$\frac{2}{\frac{1}{precision} + \frac{1}{recall}}$

MUESTRAS DESBALANCEADAS

En los casos en donde las muestras posean muy desbalanceadas la cantidad de obervaciones de cada clase, las métricas expuestas pueden distorsionarse.

Algunos tips

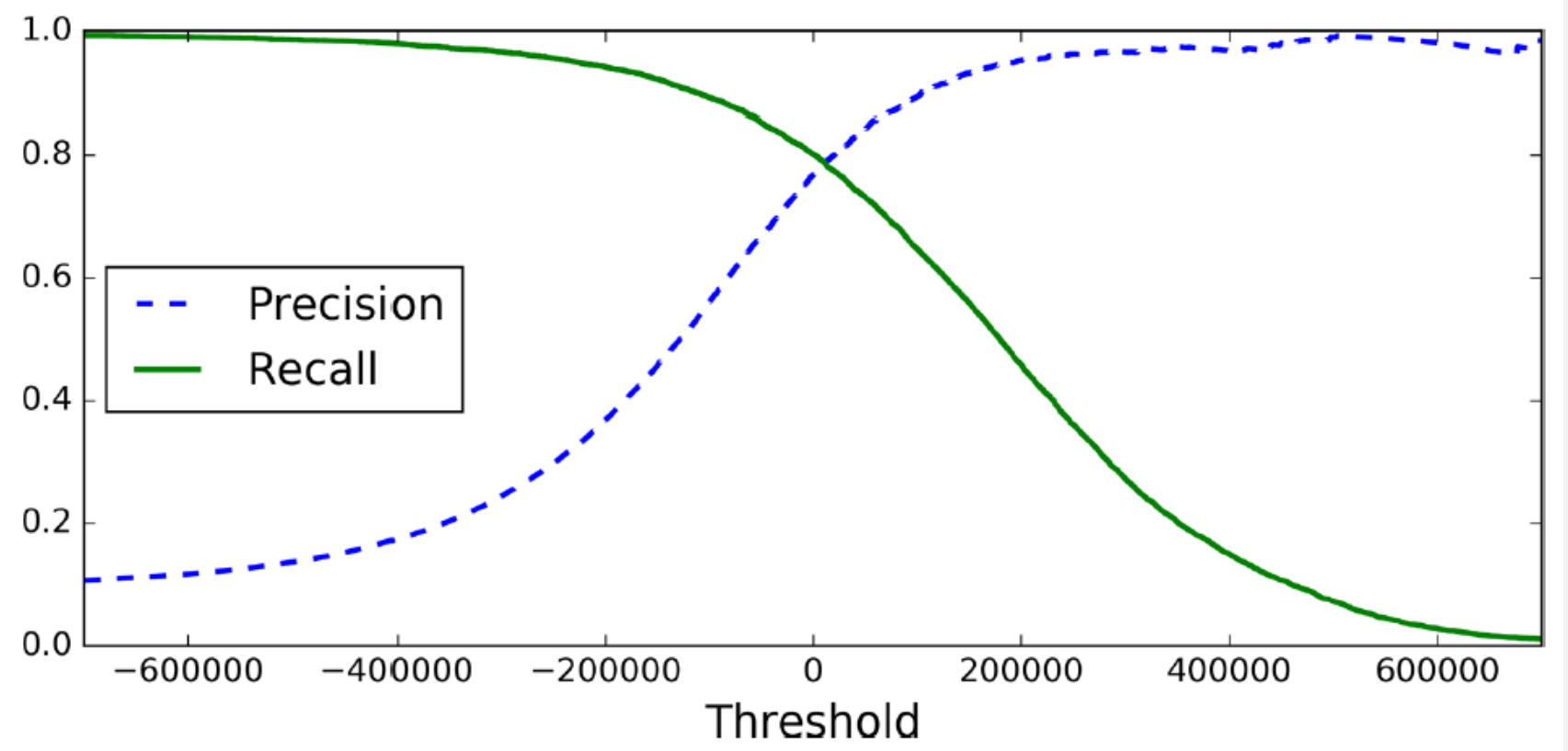
Podemos utilizar el **Matthew's Correlation Coeficient** que es una medida de performance que contempla desbalances.

`sklearn.metrics.matthews_corrcoef`

Podemos utilizar **Downsampling**, que consiste en mantener las n observaciones de la clase mas chica y de las m $(>> n)$ observaciones de la clase más grande, se extraen $\sim n$ observaciones en forma aleatoria sin reposición.

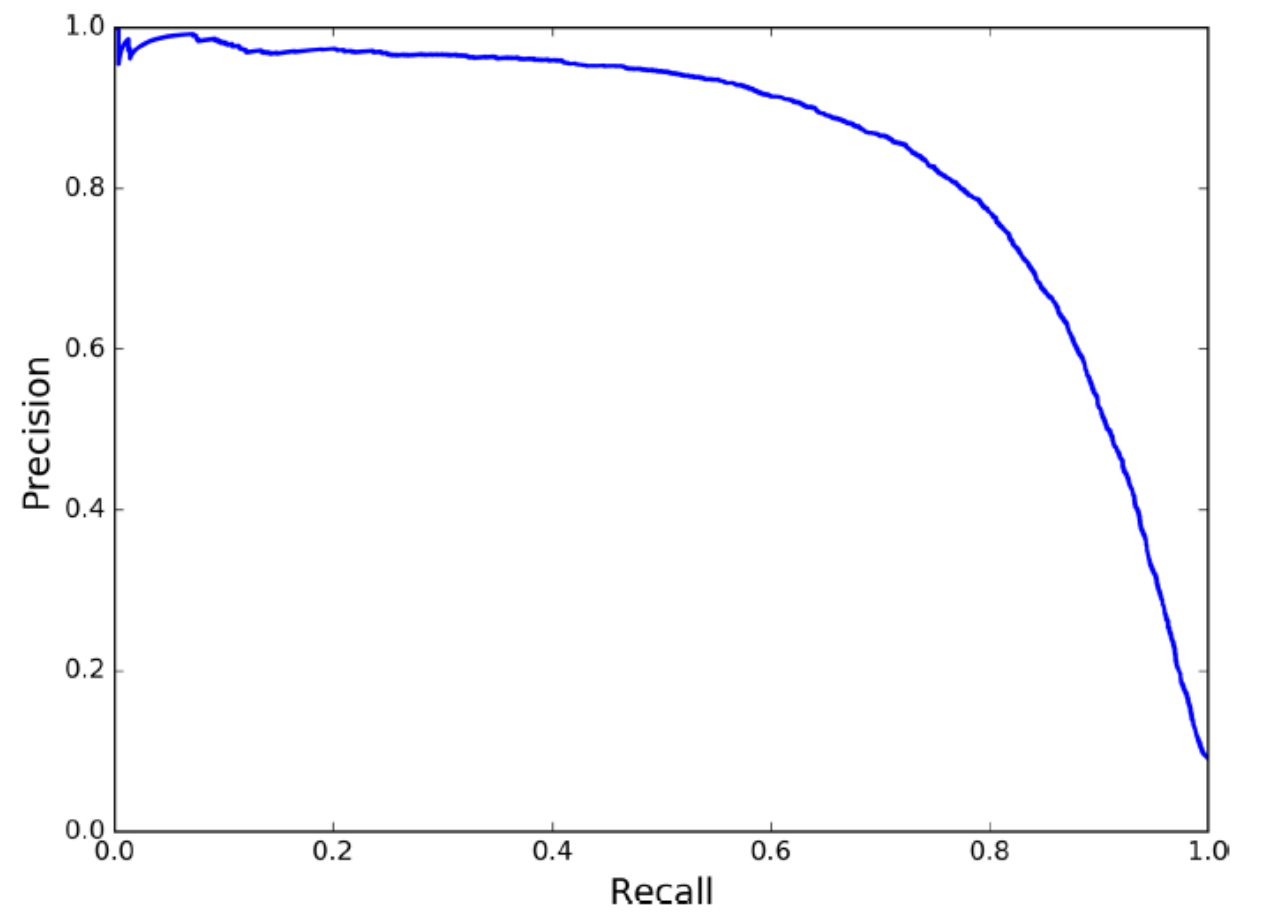
TRADE-OFF PRECISION-RECALL

Cambiar el threshold para mejorar la **precision** empeora el **recall** y vice versa.



CURVA PRECISION-RECALL

Otra forma de ver el trade-off precision-recall.



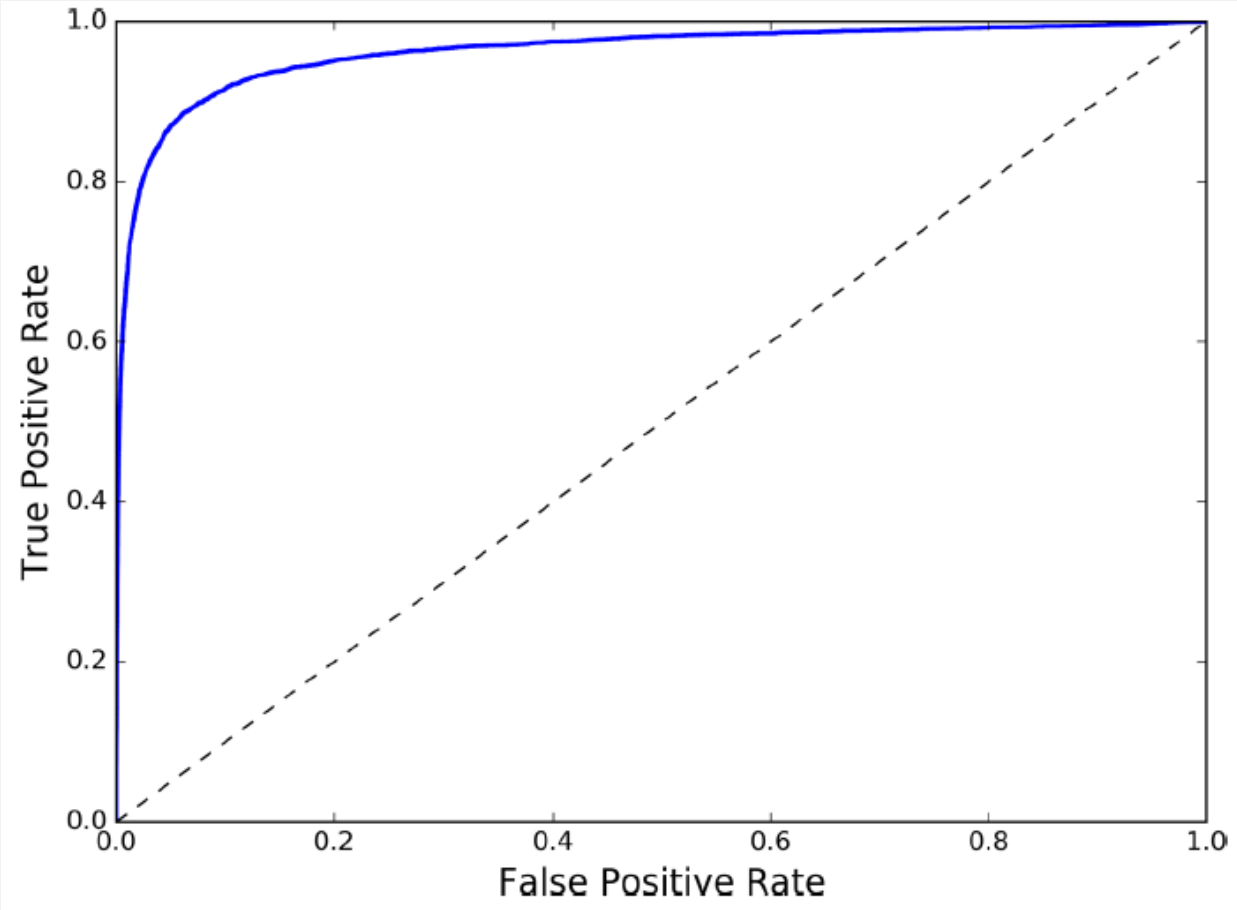
CURVA ROC

Refleja otro trade-off similar al de precision-recall.

Es uno de los más utilizados

Se puede construir un indicador llamado **AUC** (Area Under the Curve) que resume la curva.

AUC = 1	Clasificador Perfecto
AUC = 0,5	Clasificador Aleatorio



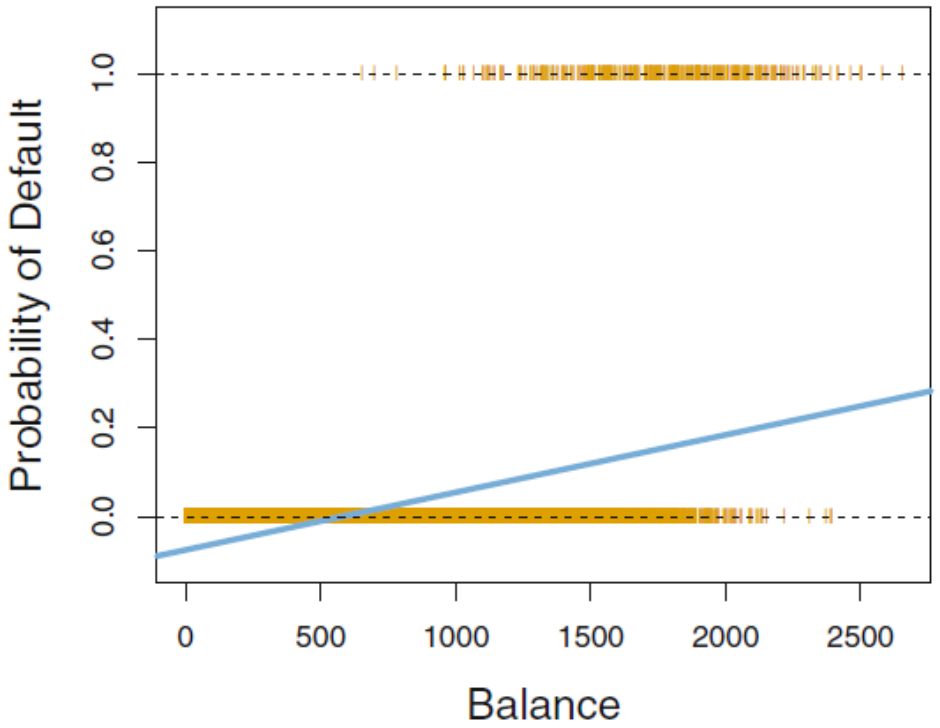
LOGISTIC REGRESSION

Modelo	$\hat{y} = \frac{e^{\beta_0 + \sum_{j=1}^{p} \beta_j \cdot x_j}}{1 + e^{\beta_0 + \sum_{j=1}^{p} \beta_j \cdot x_j}}$
Loss Function	Likelihood function (+ shrinking)

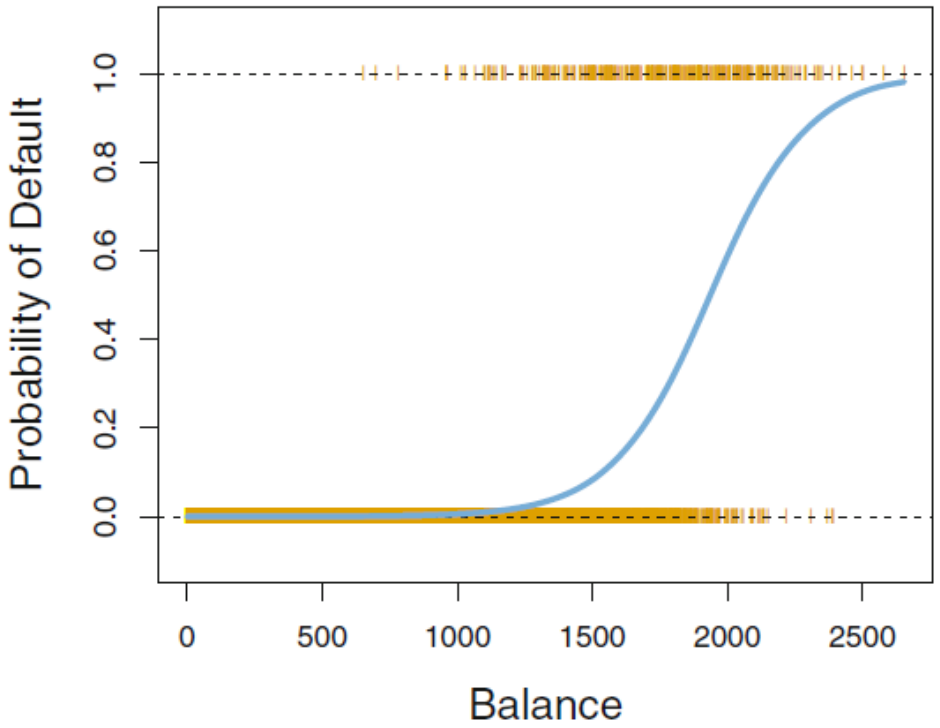
Clases relacionadas en `sklearn`
`linear_model.LogisticRegression` (**default**: penalización l2)
`linear_model.LogisticRegressionCV`

LOGISTIC REGRESSION EN ACCIÓN

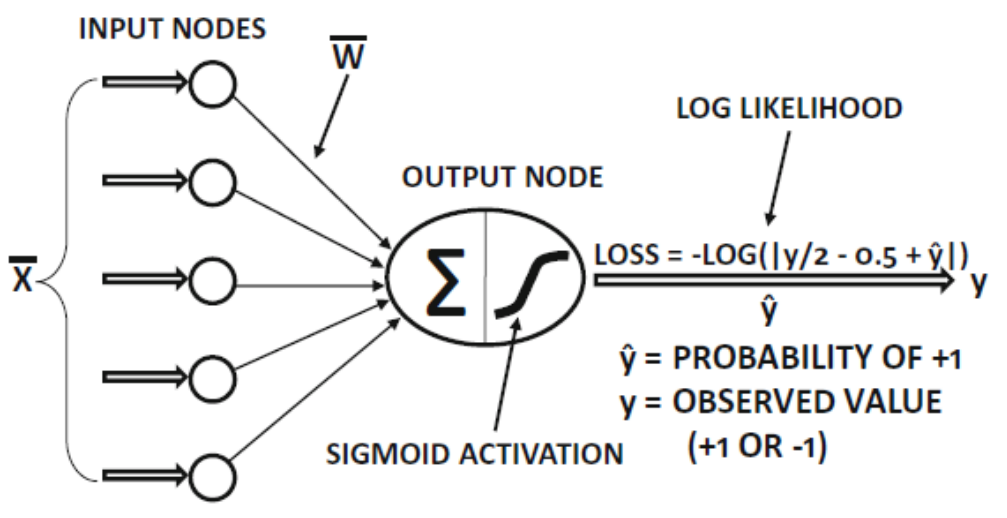
Linear Regression

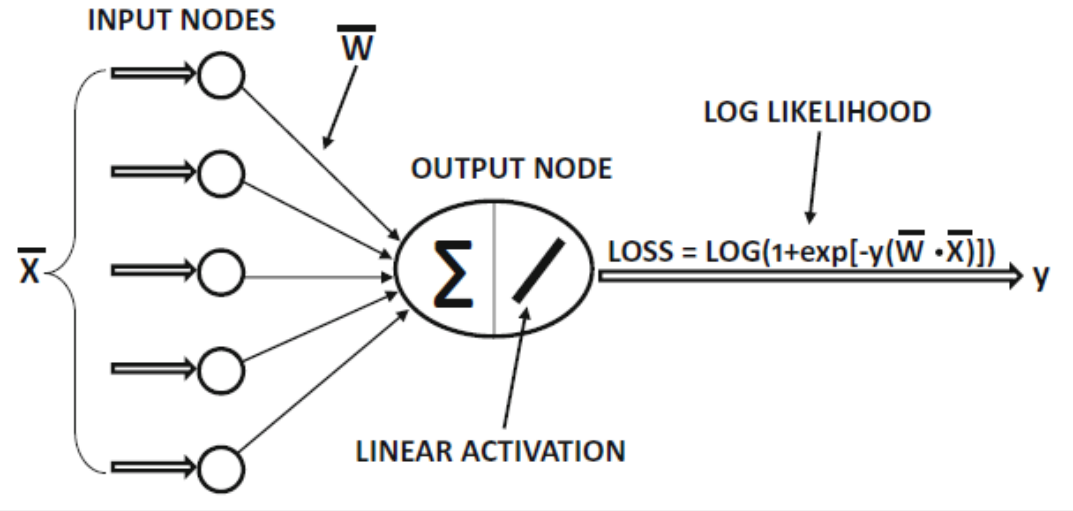


Logistic Regression



LOGISTIC REGRESSION: NN

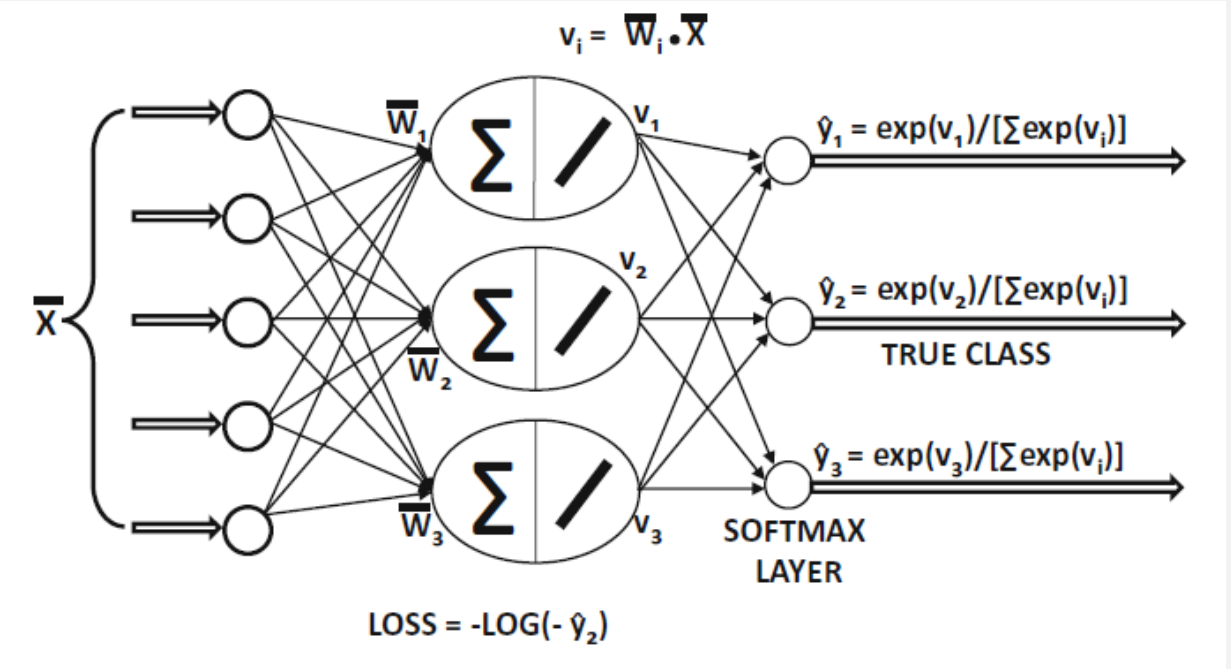




Clases relacionadas en `tensorflow`

```
keras.layers.Dense(activation='sigmoid')
```

LOGISTIC REGRESSION MULTINOMIAL: NN



Clases relacionadas en `tensorflow`
`keras.layers.Softmax`

NAÏVE BAYES

Modelo	$\hat{y} = \arg\max_y P(y) \prod_{j=1}^{p} P(x_j \mid y)$
Loss Function	Likelihood function

Naïve Bayes supone que los features son i.i.d y utiliza estimación **MAP** (Maximum a Posteriori). Es un clasificador básico pero rápido.

Lo que diferencia a cada sub-modelo entre sí, determinando cada Likelihood function particular, es la distribución que se assume $P(x_j \mid y)$.

Módulo relacionado en `sklearn`
`naive_bayes`

K-NEAREST NEIGHBOR

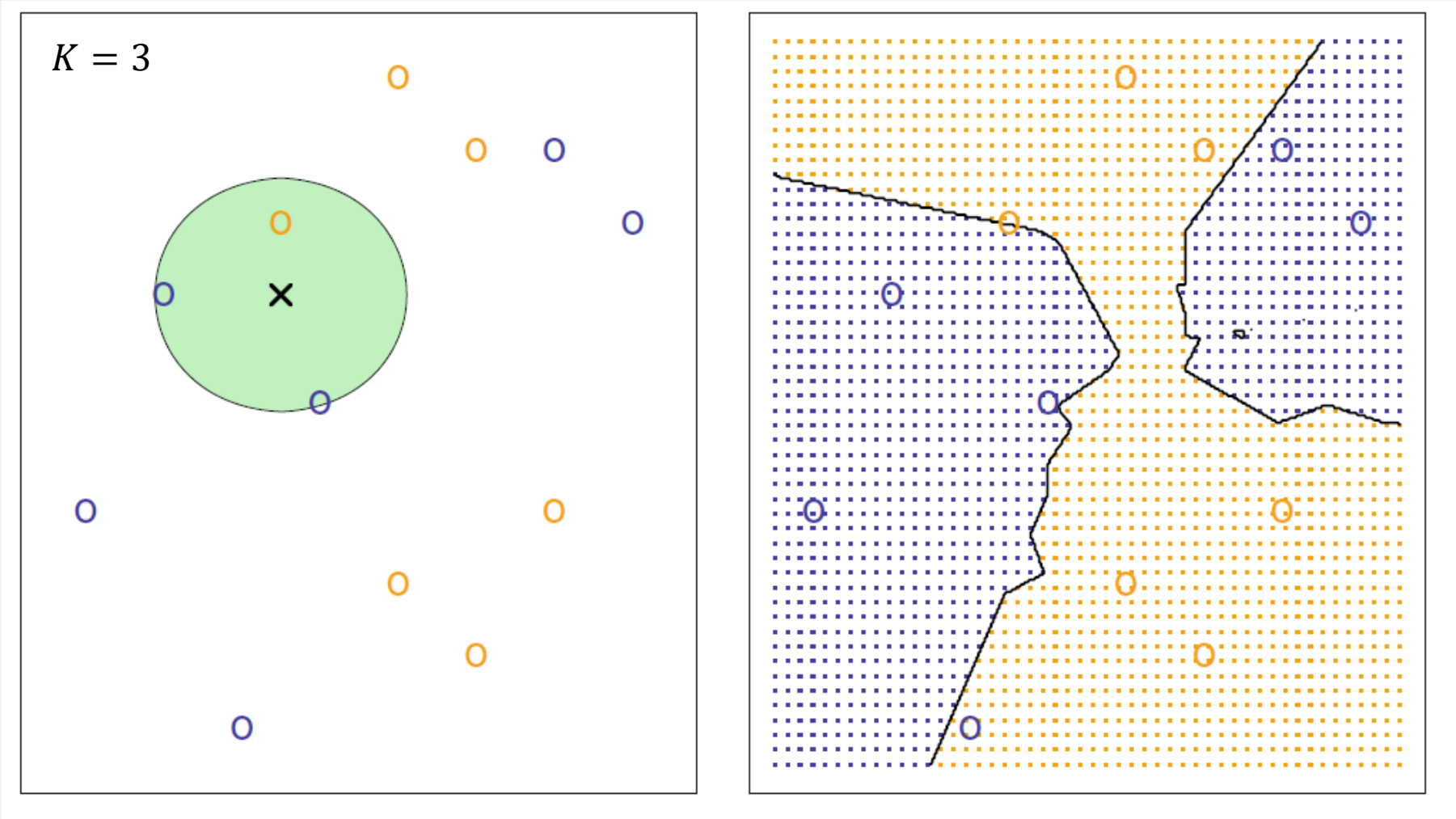
Modelo	$\hat{y} = \arg\max_y \frac{1}{K} \sum_{j \in N_{x_i}} I(y_j = y)$

Donde N_{x_i} representa el entorno de K puntos más cercanos a x_i. La cercanía estará definida por una métrica especificada.

La cantidad K suele ser estimada por cross-validation contra **Error Rate**.

Clase relacionada en `sklearn`
`neighbors.KNeighborsClassifier`

K-NEAREST NEIGHBOR EN ACCIÓN



SUPPORT VECTOR MACHINE CLASSIFIER

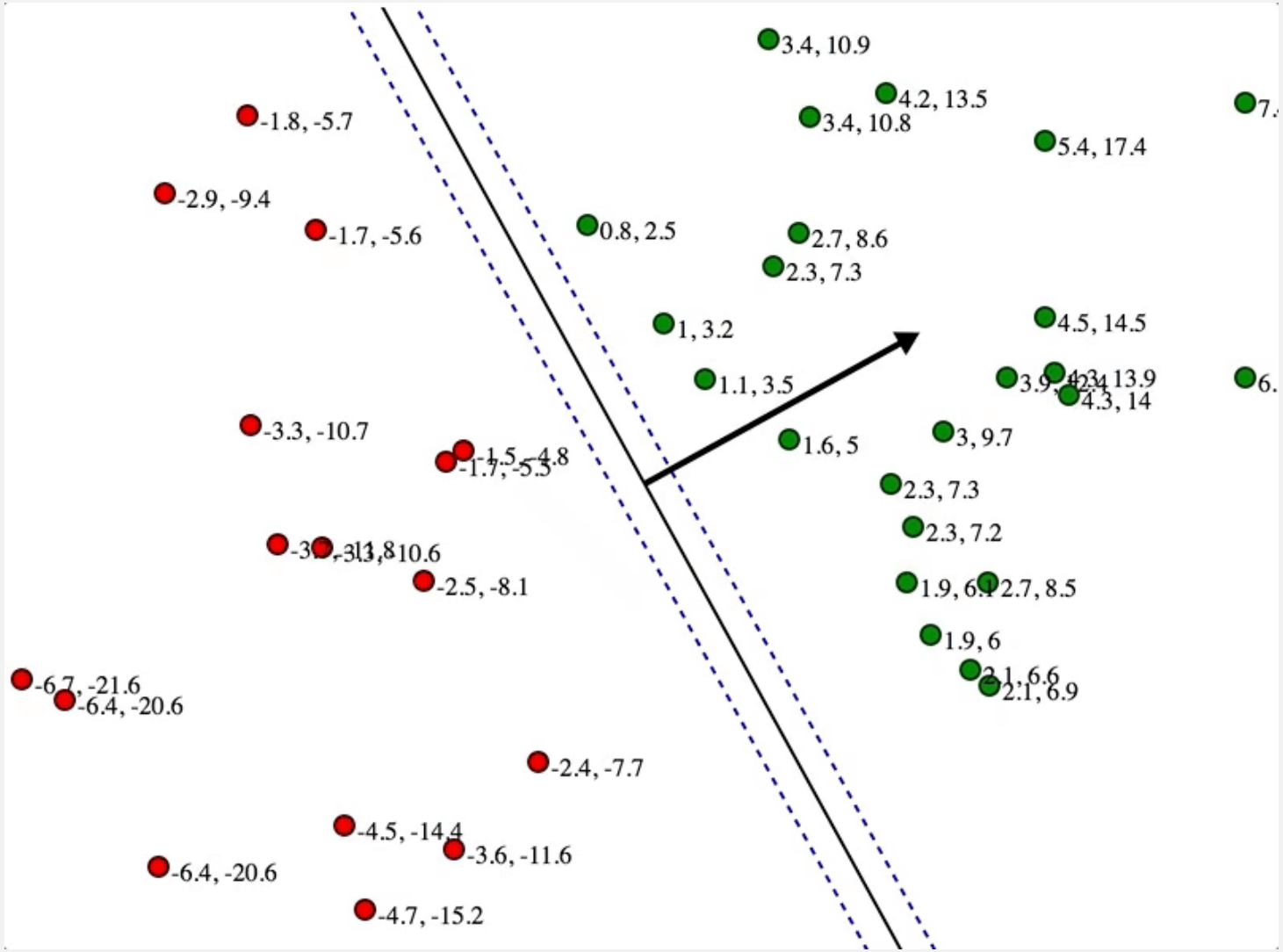
SVC busca encontrar un hiperplano divisor entre las distintas clases que **maximice** la distancia de éste con las observaciones más cercanas utilizando **soft-margins** para contemplar errores.

Modelo	$\hat{y} = sign(\beta_0 + \sum_{j=1}^{p} \sum_{m=1}^{M} \beta_{j,m} \cdot h_m(x_j))$
Loss Function	$\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \lvert y_i - \hat{y} \rvert$ (+ shrinking)

Bajos los mismos términos que SVR, aplica el uso **kernels**.

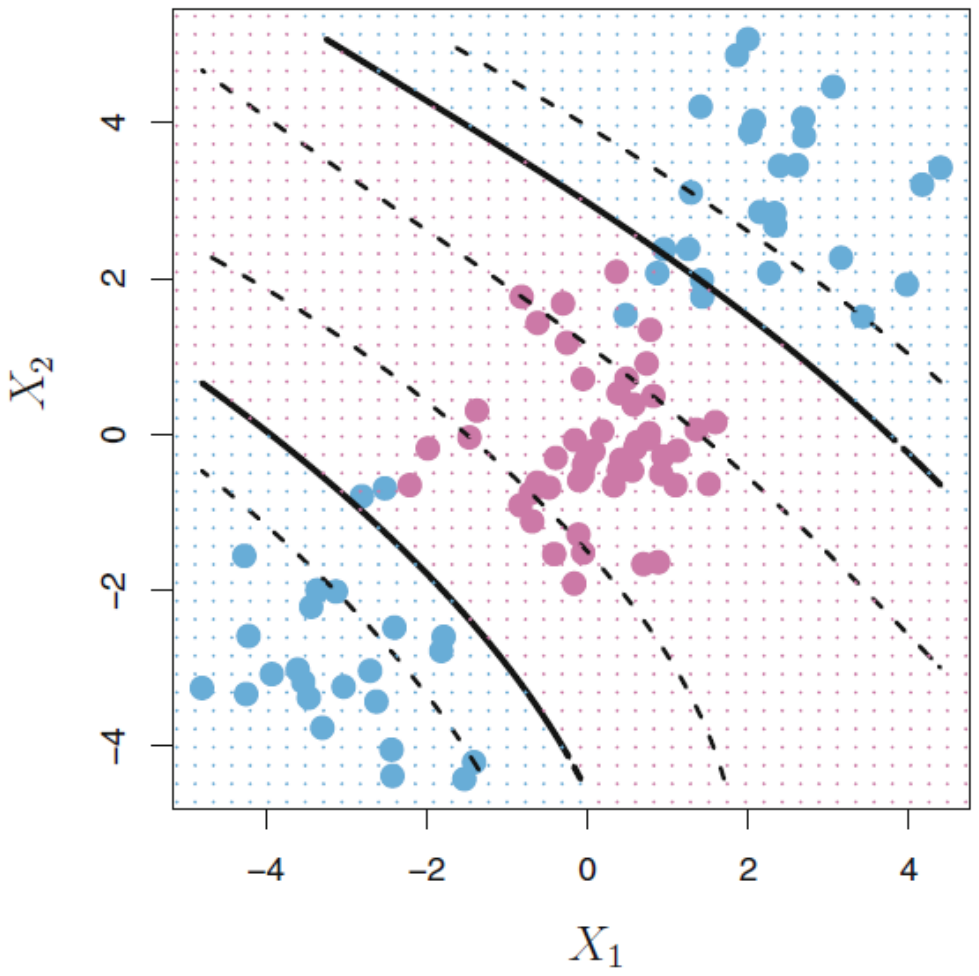
Clases relacionadas en `sklearn`	
`svm.LinearSVC`	`svm.SVC`

SUPPORT VECTOR MACHINE CLASSIFIER EN ACCIÓN

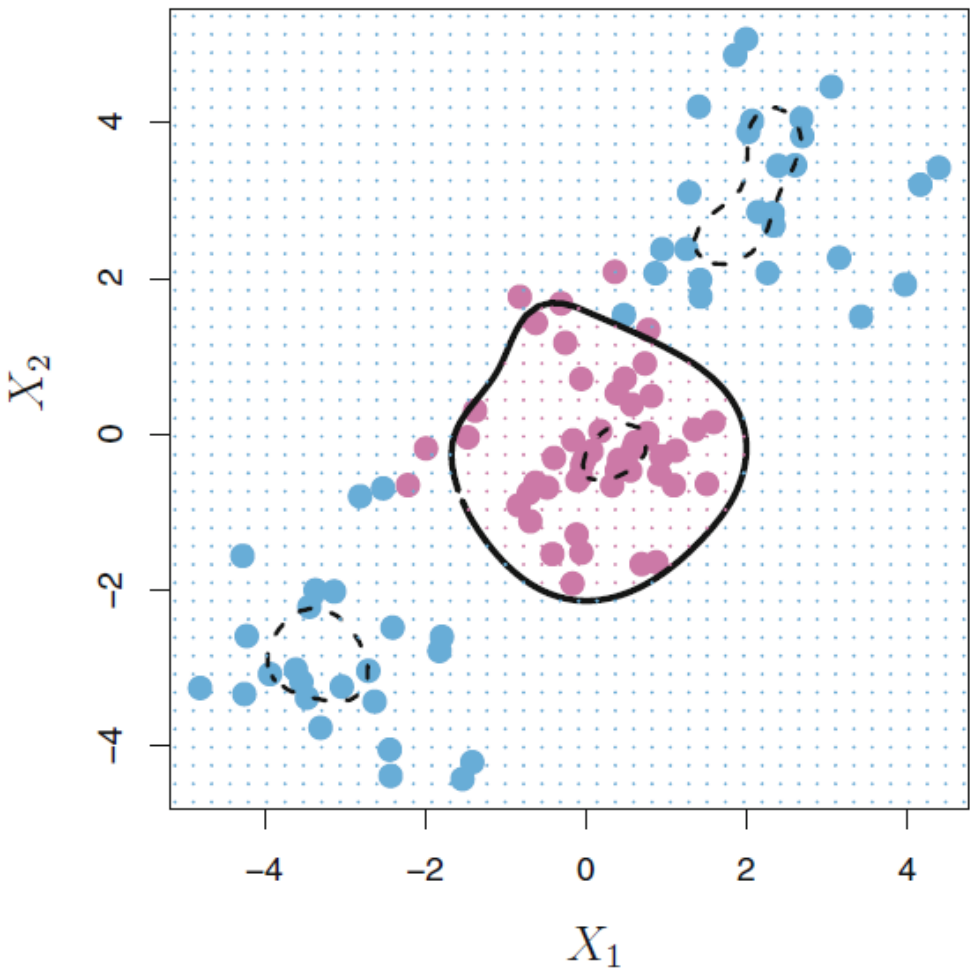


SUPPORT VECTOR MACHINE CLASSIFIER KERNEL TRICK

Kernel = polinomio de 3 grado

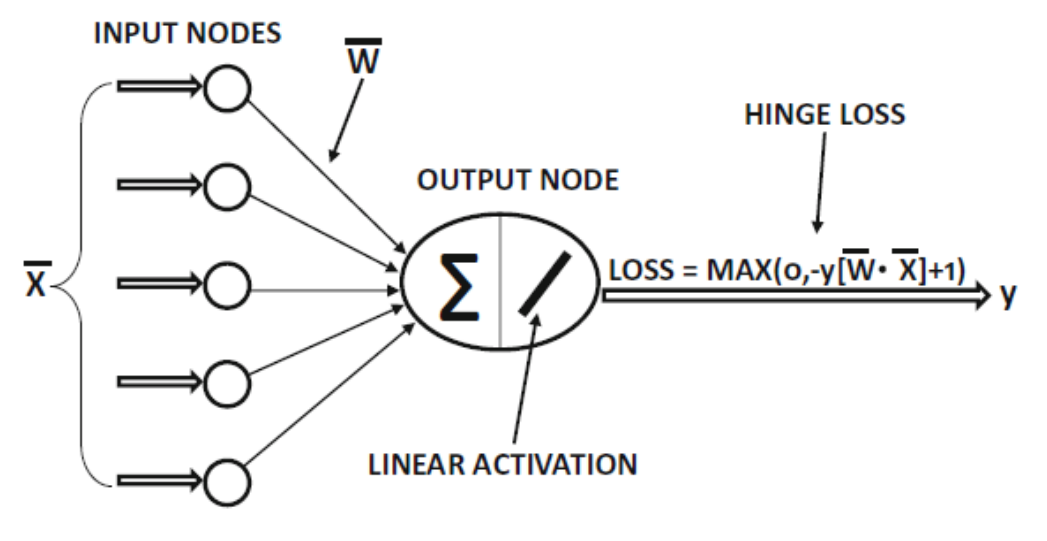


Kernel = Gaussian RBF

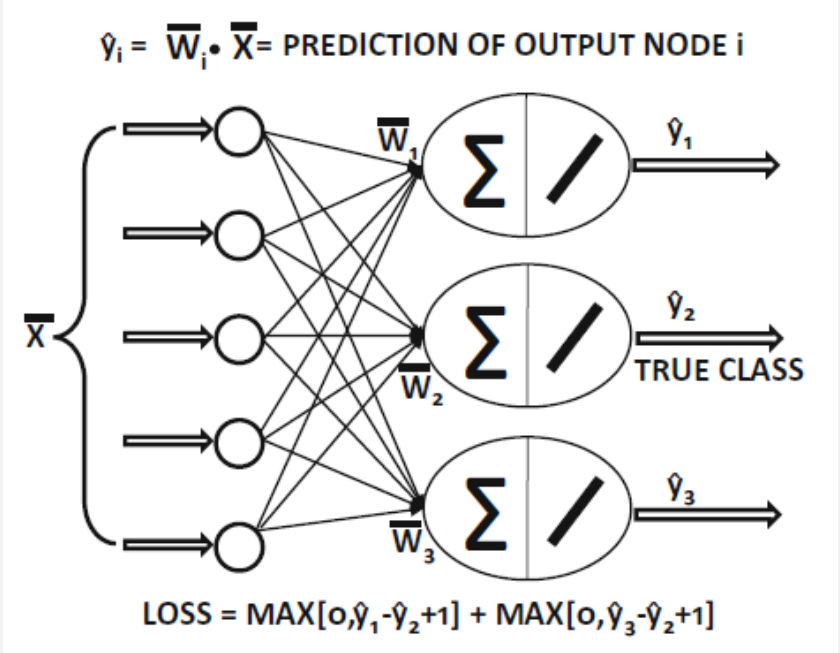


SUPPORT VECTOR MACHINE CLASSIFIER: NN

Binomial



Multinomial



Clases relacionadas en `tensorflow`

`keras.losses.Hinge`

CLASSIFICATION TREE

Replica los conceptos de Regression Tree pero con los siguientes cambios

Algoritmo General

1) Se divide el espacio de los predictores en J regiones no solapadas $R_1, R_2, \dots, R_J$ de forma greedy (la elección de las divisiones responde a **maximizar la pureza** de las ramas resultantes).
2) La predicción para una caso que cae en la region R_j es igual al **clase más común** de entre las observaciones de R_j

La pureza de R_j dadas K clases se suele medir por dos criterios

Gini	$\sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$

Entropy	$-\sum_{k=1}^{K} \hat{p}_{jk} log(\hat{p}_{jk})$

donde $\hat{p}_{jk}$ es la proporción de observaciones de la clase k en R_j

CLASSIFICATION TREE EN SCIKIT-LEARN

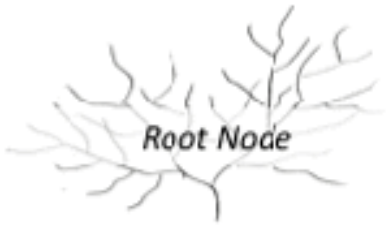
Clases relacionadas en `sklearn`
`tree.DecisionTreeClassifier`

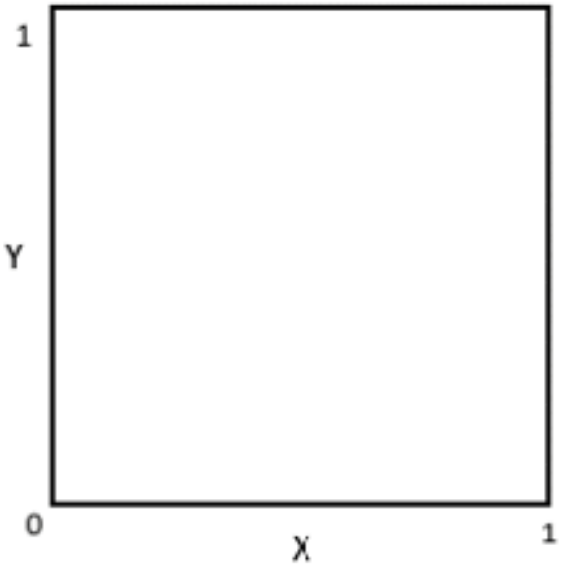
Para calibrar los hiperparámetros utilizar `optuna` o:

`sklearn.model_selection.GridSearchCV`

`sklearn.model_selection.RandomizedSearchCV`

CLASSIFICATION TREE EN ACCIÓN





Ensemble Methods

VOTING

Consiste en usar diferentes modelos en paralelo sobre el mismo dataset.

> En ensemble learning se suelen utilizar como funciones de agregación para clasificación la **moda** y para regression el **promedio**.

Para el caso específico de clasificación, el voting puede ser:

SOFT Si los modelos estiman probabilidades, se calcula el promedio.

HARD Si los modelo **no** estiman probabilidades, se calcula la moda.

El soft voting suele dar mejores resultados dado que al contemplar todo el rango de la probabilidad captura mejor clasificaciones más y menos fuertes.

Clases relacionadas en `sklearn`	
`ensemble.VotingClassifier`	`ensemble.VotingRegressor`

MÉTODOS CON SUBMUESTRAS ALEATORIAS

Consisten en usar modelos idénticos en paralelo sobre distintas submuestras obtenidas aleatoriamente del dataset original.

BAGGING	Se muestrea sobre **todos** los regresores **con reposición**.
PASTING	Se muestrea sobre **todos** los regresores **sin reposición**.
RANDOM SUBSPACES	Se muestrean los **features a utilizar**, dejando **todas** las observaciones de cada uno.
RANDOM PATCHES	Se muestrean los **features a utilizar**, y **también** se muestrea sobre las observaciones de cada uno.

BAGGING: PARTICULARIDADES

Clases relacionadas en `sklearn`
`ensemble.BaggingClassifier`
`ensemble.BaggingRegressor`

Dada la forma particular de muestreo que utiliza bagging, surge una técnica de validación asociada muy útil que se denomina **out-of-bag validation**.

Ésta consiste en utilizar como validation set para cada corrida las observaciones que quedaron fuera de la selección del resampling.

RANDOM FOREST

Usa específicamente **Decision Trees**, y busca generar árboles lo más descorrelacionados posibles entre sí, combinando:

a) bagging, y

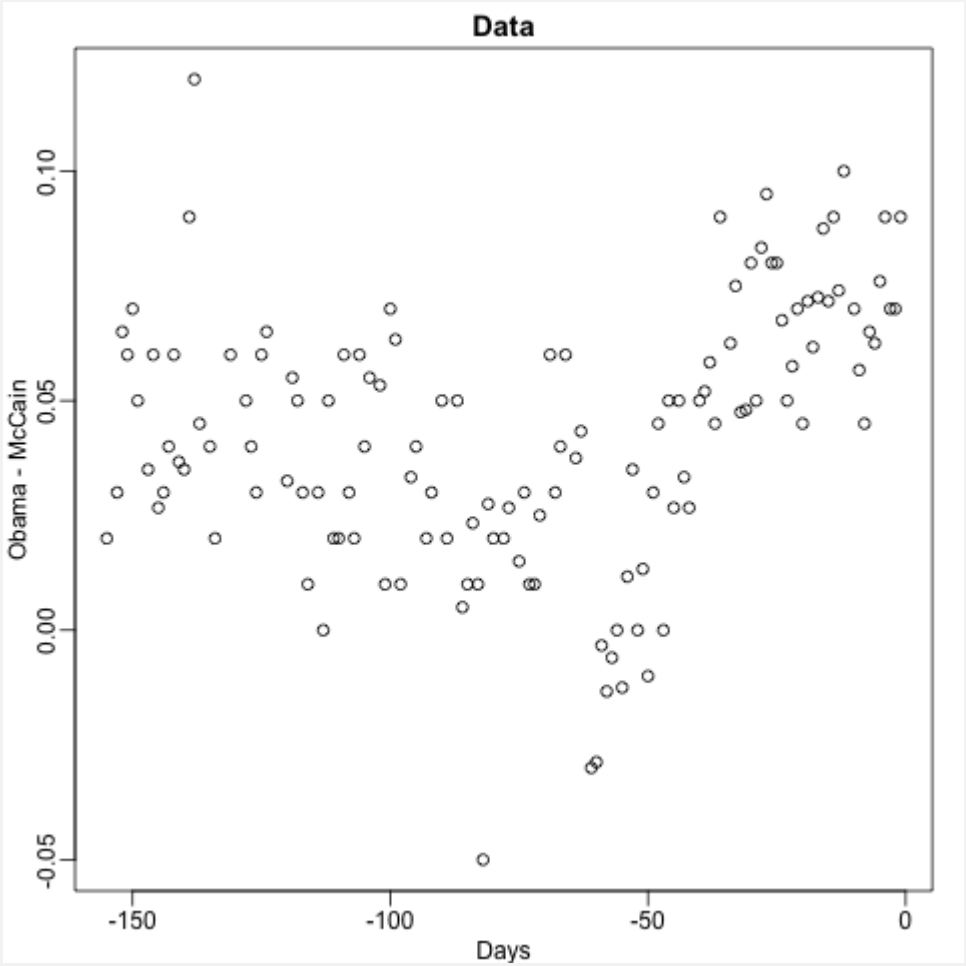
b) ante cada decisión de split, selecciona sobre m f**eatures** al azar

> La clave de romper la correlación entre árboles radica en fijar m mucho más pequeño que p (usualmente $m = \sqrt{p}$) lo que evita que features muy importantes estén siempre en las submuestras.

Son muy buenos para análisis de **Feature Importance**.

Clases relacionadas en `sklearn`	
`ensemble.RandomForestClassifier`	`ensemble.RandomForestRegressor`

RANDOM FOREST EN ACCIÓN



BOOSTING

Consiste en usar modelos idénticos en forma **secuencial**, entrenando uno sobre la información del previo.

El caso más usual es con **Decision Trees**.

El algoritmo general de entrenamiento consiste de

1. Entrenar un árbol
2. Incorporar el árbol entrenado a la función de aproximación que consolida aditivamente a todos los árboles.
3. Transformar el dataset incorporando la información del árbol entrenado y retroalimentar en 1.

Al existir dependencia de los árboles entrenados en pasos previos suele ser difícil de paralelizar y genera soluciones **path dependent**.

BOOSTING: HIPERPARÁMETROS

El método general suele poseer 3 hiperparámetros:

NÚMERO DE ÁRBOLES A ENTRENAR

Dada la naturaleza secuencial puede producirse overfitting.

PROFUNDIDAD DEL ÁRBOL

Usualmente baja (1 a 3 splits), dado que la clave está en ir mejorando la aproximación combinando pequeños árboles.

VELOCIDAD DE APRENDIZAJE

Cuanto más lento el aprendizaje (parámetro más bajo) se puede incorporar más flexibilidad, pero requiere mayor cantidad de árboles.

ADABOOST Y GRADIENT BOOSTING

Se diferencian en cómo aplican el **punto 3** del algoritmo general.

ADABOOST

Busca mejorar la aproximación mediante: **a)** aumento de la ponderación de los datos donde mayor error se está cometiendo, y **b)** aumento de la ponderación de los mejores árboles en la función de aproximación.

GRADIENT BOOSTING

Busca mejorar la aproximación ajustando cada árbol sobre los residuos del árbol previo.

Clases relacionadas	`sklearn`	`ensemble.AdaBoostClassifier`
		`ensemble.AdaBoostRegressor`
	`xgboost`	`XGBRegressor / XGBClassifier`

STACKING Y OTROS ENSEMBLES

Consiste en usar n capas secuenciales de m modelos en paralelo con un modelo final mezclador (**blender**) entrenado a partir del output de la última de las n capas anteriores.

Para el entrenamiento los datos se dividen en $n + 1$ tramos, de forma de tener datos para testing para cada capa más el blender.

Librería relacionada
`https://github.com/scikit-learn-contrib/DESlib`

La librería `DESlib` posee además algoritmos basados en **Dynamic Selection** que buscan encontrar el mejor modelo o ensemble durante la etapa de testing.

El problema de Clustering

CONCEPTO GENERAL

El problema de **clustering** consiste en encontrar la forma de poder construir subgrupos o **clusters** de los datos en forma **no supervisada**.

> La clave del clustering es agrupar la información de manera de que los datos contenidos dentro de un grupo sean **similares** entre sí, al tiempo que comparados con datos de otros grupos sean lo más **distintos** posible.

Como suele ocurrir en otros métodos, esa similaridad estará dada por el ámbito de aplicación correspondiente.

Estos métodos buscan dar información sobre los datos, **no predicciones**, pero pueden combinarse con clasificación una vez clusterizados los datos.

> Asimismo no son muy robustos a **perturbaciones** en los datos.

K-MEANS

Busca particionar **todas** las observaciones en K clusters **sin intersección**.

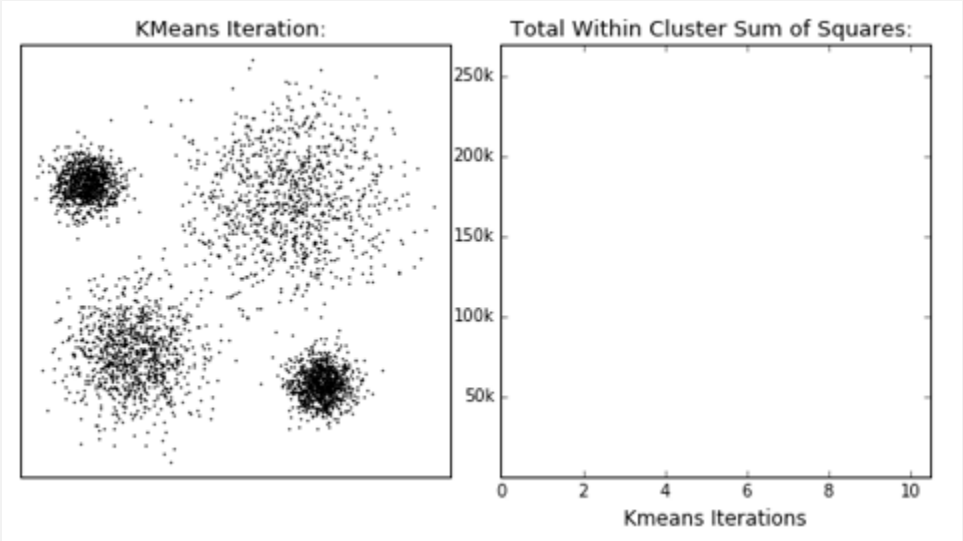
Loss Function	$\sum_{k=1}^{K} W(C_k)$

Donde C_k es el cluster k, y W es la métrica de diferencia interna del mismo.

El algoritmo numérico usado habitualmente halla **mínimos locales**, por lo que es sensible a las **ubicaciones iniciales de los centroides** (que se generan aleatoriamente). Por ello, la estimación se suele repetir varias veces y se selecciona la solución con menor Loss Function.

Una de las dificultades de éste método es qué cantidad K de clusters utilizar.

K-MEANS EN ACCIÓN



Clases relacionadas en `sklearn`

`cluster.KMeans`

HIERARCHICAL CLUSTERING

Busca construir una representación de los datos basada en árboles que se denomina **dendograma**.

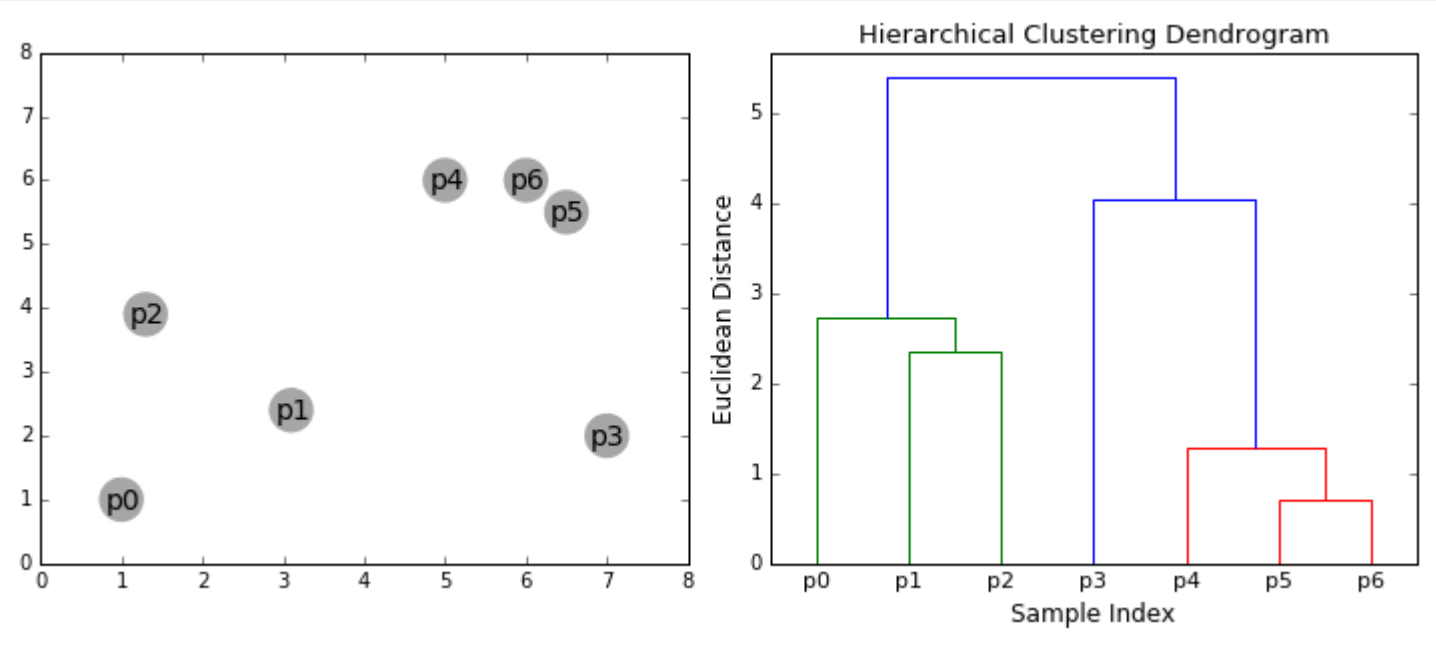
> En su variante **Agglomerative**, este dendograma se construye desde abajo hacia arriba, partiendo de cada observación como una hoja para ir uniéndose en ramas a partir de la **similaridad**.
> Cuanto **más abajo** se unan las observaciones **más similares** son, y la altura de corte del árbol actúa como el K de K-Means.

Además de la métrica de similaridad entre observaciones (**affinity**), se suma el **linkage**, que es la métrica usada para medir la similaridad entre grupos.

Es clave en este método la **estandarización** de variables heterogéneas.

También existe otra variante, menos común, llamada **Divisive** clustering.

AGGLOMERATIVE HIERARCHICAL CLUSTERING EN ACCIÓN



Clases relacionadas en `sklearn`

`cluster.AgglomerativeClustering`

GAUSSIAN MIXTURES

Es un modelo probabilístico (**generativo**) que asume que los datos son generados por una combinación (**mixture**) de un número finitio de distribuciones **gausianas**.

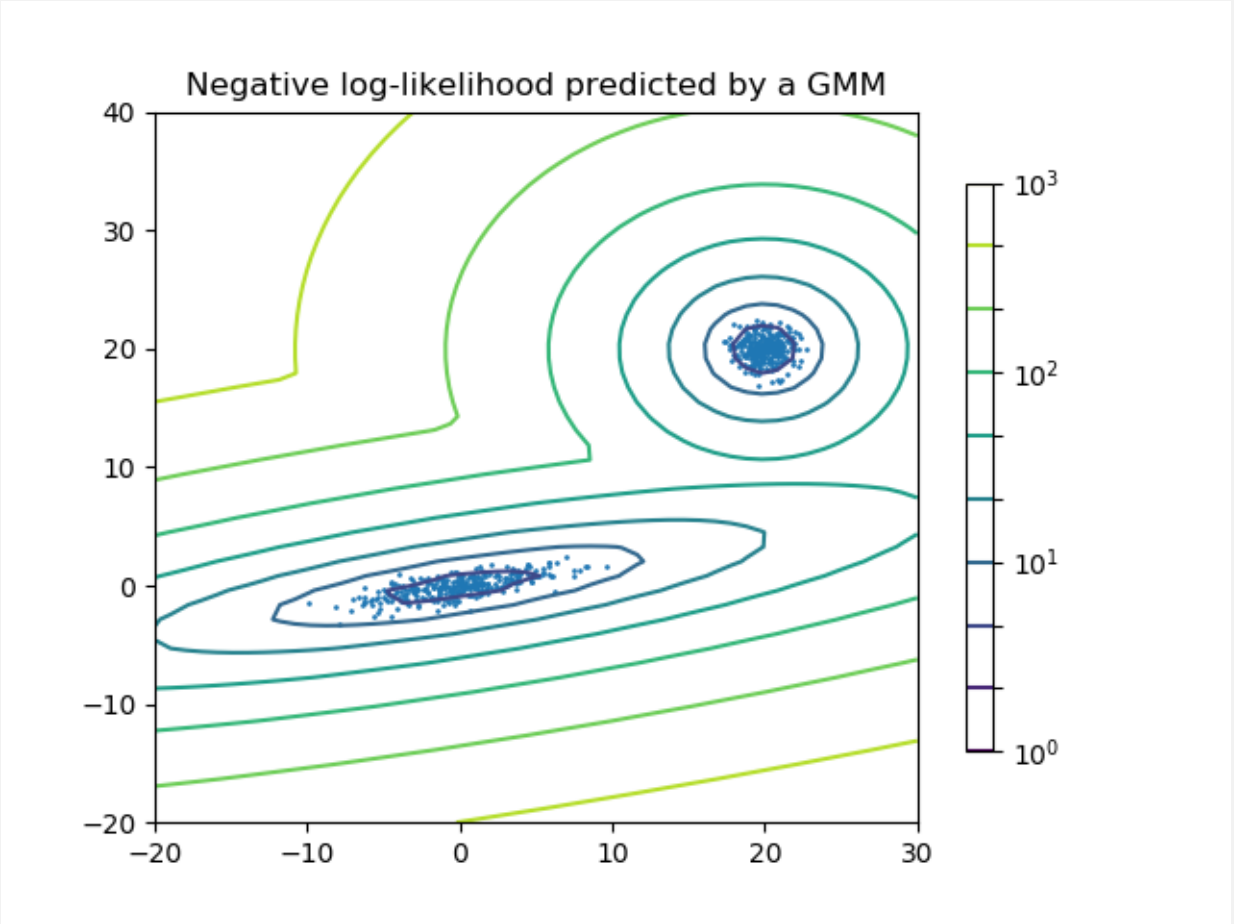
Loss Function	$p(x\|\theta) = \sum_{i=1}^{K} \pi_i \mathcal{N}(x\|\mu_i, \Sigma_i)$

Para la estimación de los parámetros se utiliza el algoritmo **EM**

Se puede considerer una version **soft** de K-Means dado el uso de distribuciones de probabilidad como modelos generadores de los clusters.

Clases relacionadas en `sklearn`
`mixture.GaussianMixture`

GAUSSIAN MIXTURES EN ACCIÓN



CANTIDAD DE CLUSTERS CORRECTA

La búsqueda de la cantidad correcta de clusters **no tiene** respuesta única y dependerá muchas veces del análisis gráfico de los datos y el uso de tests.

MÉTODO DEL CODO

Se entrena el modelo para varios K y se calcula la suma total de cuadrados dentro de cada cluster (**WSS**). A partir de graficar WSS vs. K se toma el valor de K donde **quiebre** el gráfico (codo).

Calinsky-Harabasz Score

Ratio de la dispersión media entre clusters sobre la dispersión media dentro de los clusters. A mayor valor mejor score.

scikit-learn: `sklearn.metrics.calinski_harabasz_score`

HIDDEN MARKOV MODELS Y NN

"Clustering Sequences with Hidden Markov Models"

P. Smyth

http://papers.nips.cc/paper/1217-clustering-sequences-with-hidden-markov-models.pdf

"Clustering with Deep Learning: Taxonomy and New Methods"

E. Aljalbout et al.

https://arxiv.org/abs/1801.07648

El problema de Representación

CONCEPTO GENERAL

El problema de **representación** busca encontrar una manera simplificada, o más "pura", de ver la información (ej. **reducción de dimensionalidad**).

> La idea base es que los datos subyacen en una "curva" (**manifold**) que se encuentra "mezclada" (**embedded**) en el espacio **observado**. Dicho manifold **puede** ser de menor dimensionalidad al espacio observado.

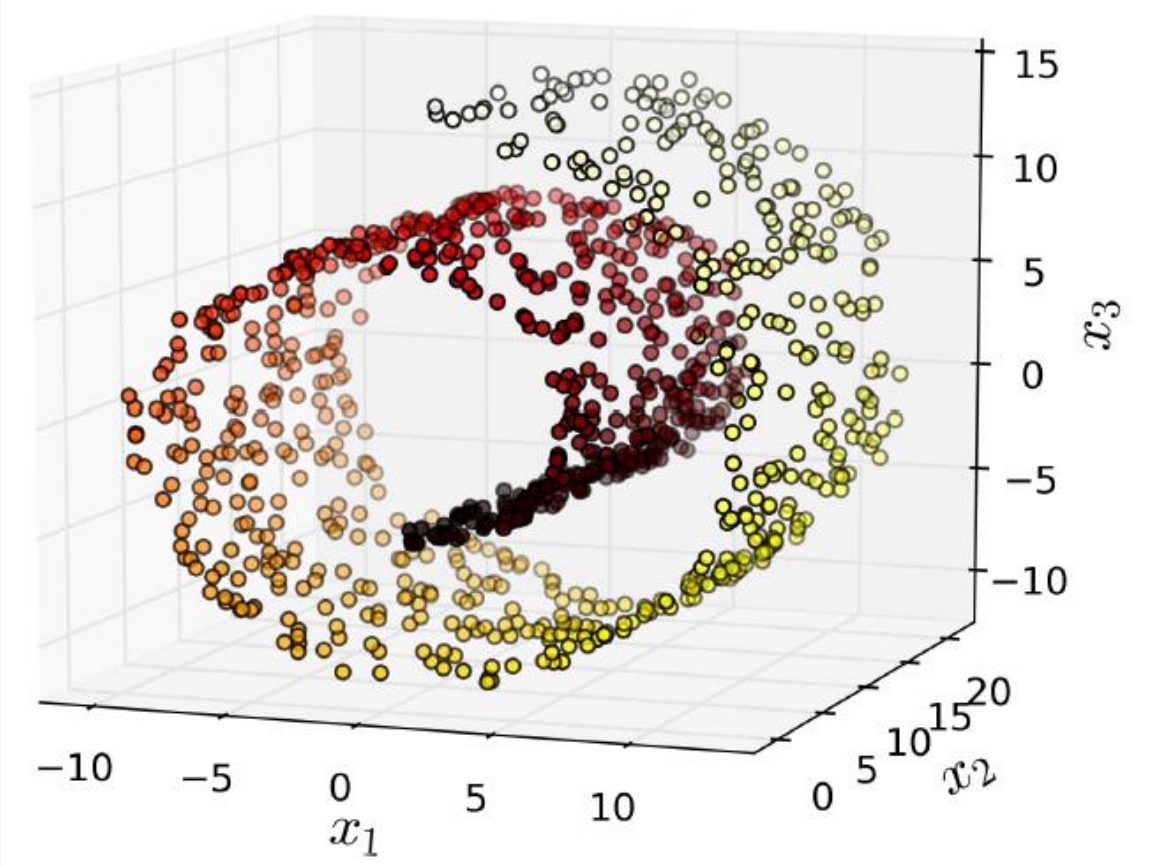
Cada modelo se diferencia en cómo plantea el manifold y su **proyección** sobre el espacio observado. Existen principalmente 2 formas generales:

VARIABLES LATENTES: Los datos observados provienen de un manifold de variables no observables (ej. **Factor Decomposition**).

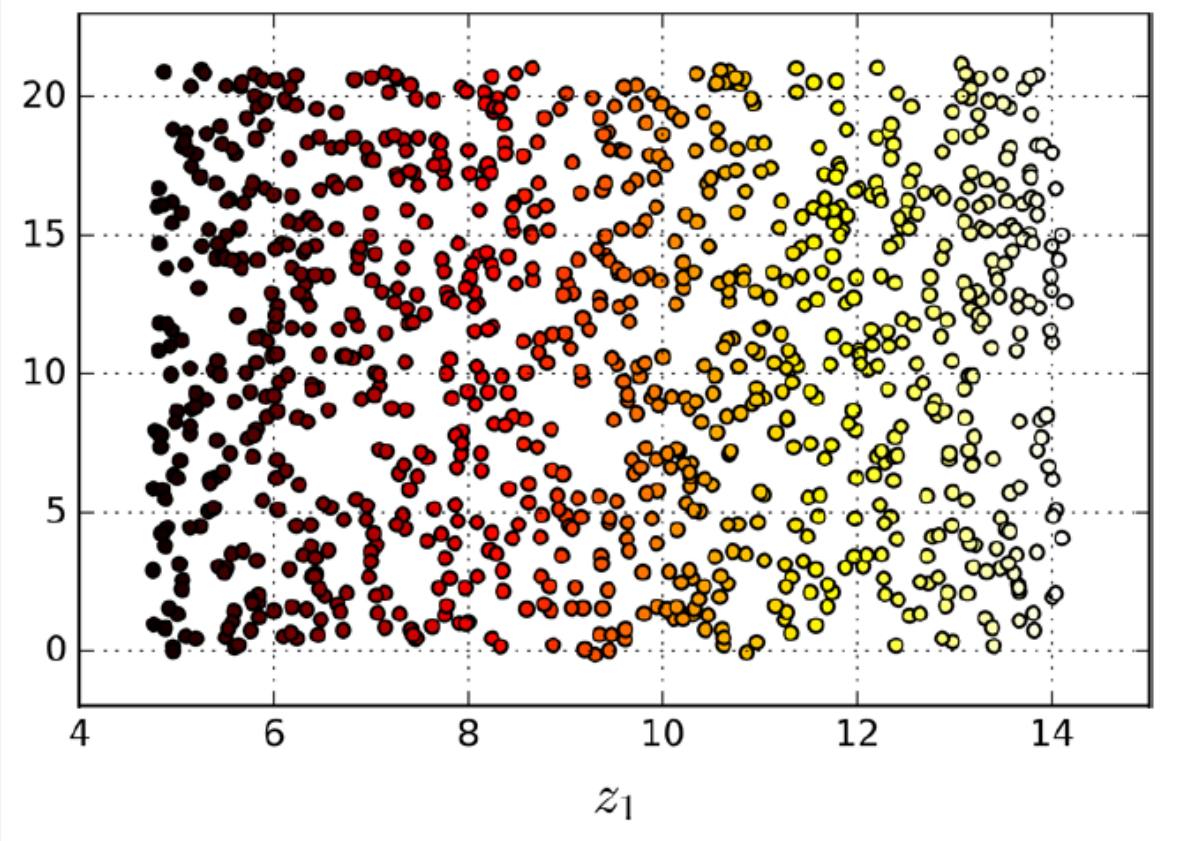
MÉTODOS GEOMÉTRICOS/TOPOLÓGICOS: La proyección se realiza a través de transformaciones (conocido como **Manifold Learning**)

PLANTEO GRÁFICO CONCEPTUAL

Espacio observado de 3 dimensiones



Manifold subyacente de 2 dimensiones



REPRESENTACIÓN Y PIPELINES

En la mayoría de los casos el output de los modelos de representación suele utilizarse como input de otros modelos de ML **preprocesando** los datos.

Esta forma de combinar modelos se denomina **pipeline**.

Lo que vuelve especialmente relevante al uso de pipelines es el hecho de pueden hacer que el modelo final de ML obtenga mejoras en su **perfomance** yendo desde su poder de generalización hasta su eficiencia computacional.

Clases relacionadas en `sklearn`
`pipeline.Pipeline`

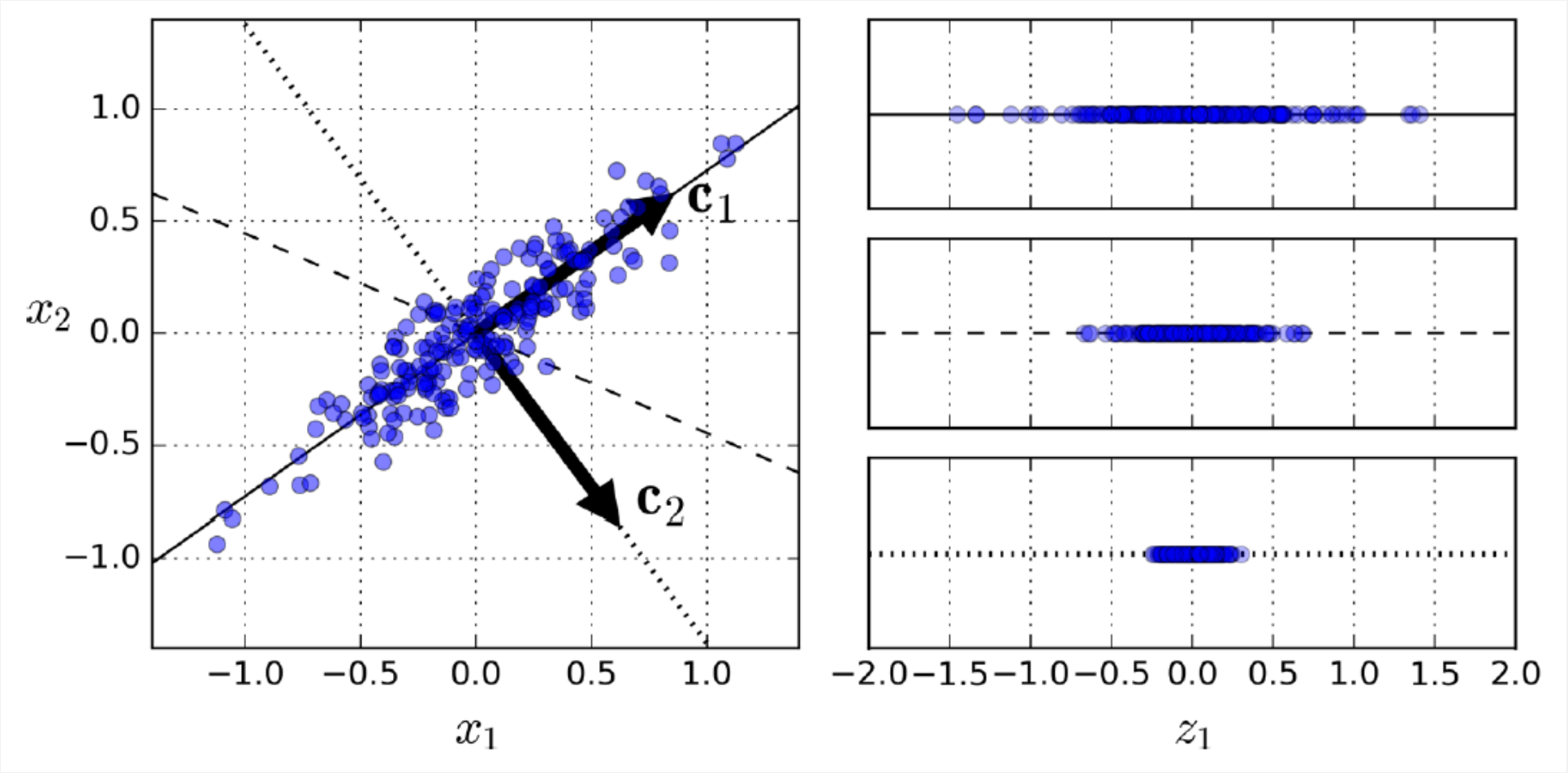
PRINCIPAL COMPONENTS ANALYSIS

Establece que los datos observados provienen del resultado de la interacción de un conjunto de variables latentes **ortogonales** (incorrelacionados), que proyectan linealmente sobre el espacio observado preservando su varianza.

Para hallar los vectores de proyección (**Componentes Principales**), una vez **centrados** los datos, se utilizan la factorización matricial de la matriz de varianzas y covarianzas en Autovalores y Autovectores (**Eigendecomposition**), o la **Singular Value Decomposition**.

Clases relacionadas en `sklearn`
`decomposition.PCA`

PRINCIPAL COMPONENTS ANALYSIS EN ACCIÓN



PRINCIPAL COMPONENTS ANALYSIS PARA COMPRESIÓN

PCA permite conocer la **porción de la varianza** total que explica cada CP. Ello resulta del cociente entre el autovalor correspondiente al CP (varianza del CP), y la suma de todos los autovalores (varianza total).

Esta característica nos otorga un criterio simple y robusto de **reducción de dimensionalidad:**

1) Se fija un umbral **mínimo** de varianza total a explicar.
2) Se selecciona el primer grupo de CP ordenados en forma descendente por porción de varianza explicada que alcance tal umbral. Al ser **incorrelacionados**, la suma de sus varianzas es la varianza total del grupo.

KERNEL PCA

Generaliza PCA para comportamientos no lineales en los datos a través del uso de **kernels** (análogamente a lo visto en Support Vector Machines)

> Kernel-PCA plantea un espacio latente (que puede ser de gran dimensionalidad) que **proyecta** en el espacio observado mediante kernels; y luego modela el manifold subyacente a dicho espacio latente como PCA.

Kernel-PCA se utiliza para obtener una representación más desagregada y detallada sobre datos de origen no lineal.

Clases relacionadas en `sklearn`
`decomposition.KernelPCA`

KERNEL PCA COMENTARIOS

A diferencia de PCA, Kernel-PCA:

1. Puede resultar en una gran cantidad de CP dada la potencial mayor dimensión del espacio latente.
2. Se pierde, por del uso de kernels, el criterio de reducción de dimensionalidad por porcentaje de varianza explicada.

Si se quiere reducir dimensionalidad con este modelo, se pueden encontrar tanto la **cantidad de CP,** como el **kernel** óptimos mediante CV utilizando **pipelines**.

INDEPENDENT COMPONENTS ANALYSIS

ICA plantea un manifold compuesto por factores latentes **estadísticamente independientes** que proyectan linealmente sobre el espacio observado.

> **PCA** es un caso particular si asumimos que los factores son **Gausianos**, dado que allí la incorrelación garantiza independencia estadística.

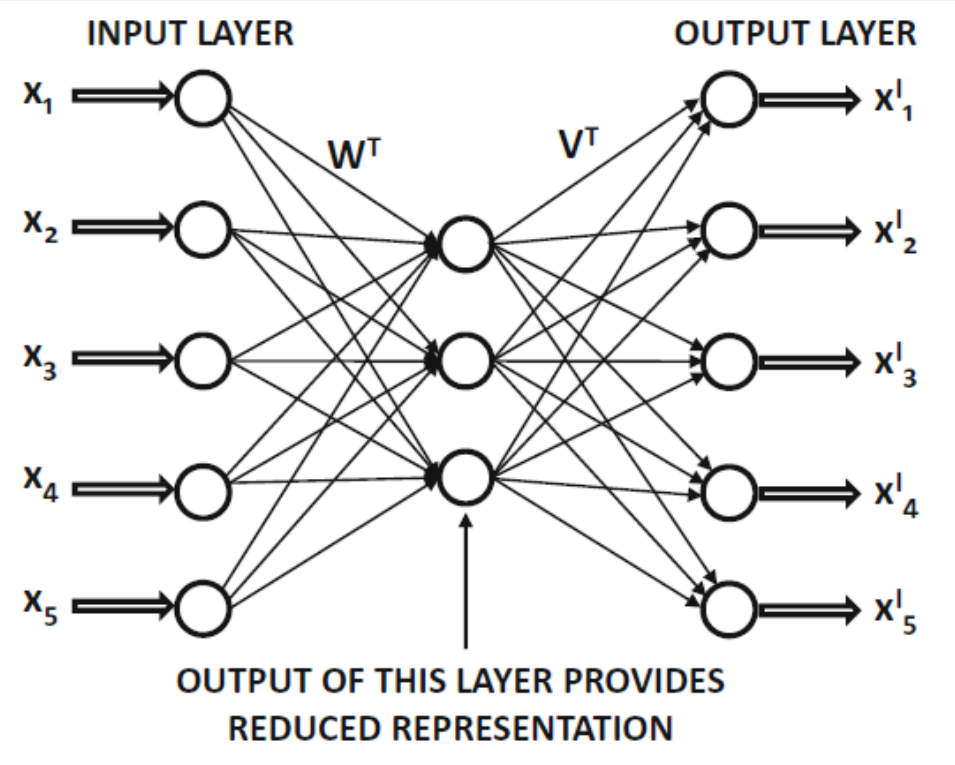
ICA se utiliza para **descomponer** una señal observada, brindándonos una nueva representación de los datos en componentes independientes.

La solución de este problema se consigue a partir de la minimización de la suma de las **entropías** de cada componente.

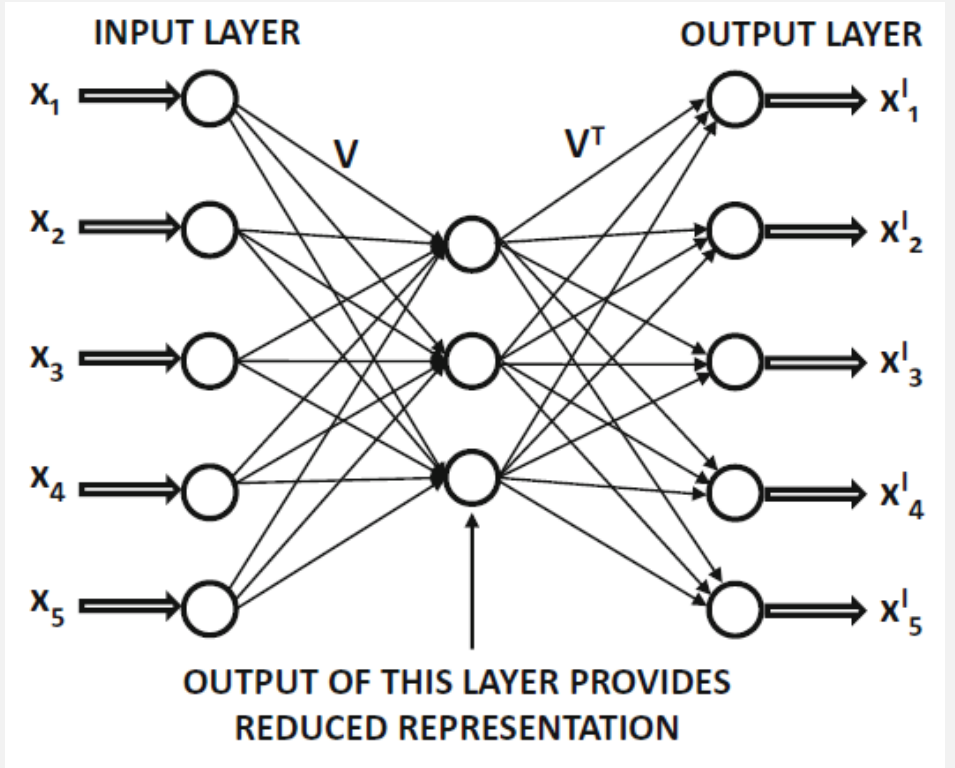
Clases relacionadas en `sklearn`
`decomposition.FastICA`

AUTOENCODER LINEAL

Free Weights



Tied Weights

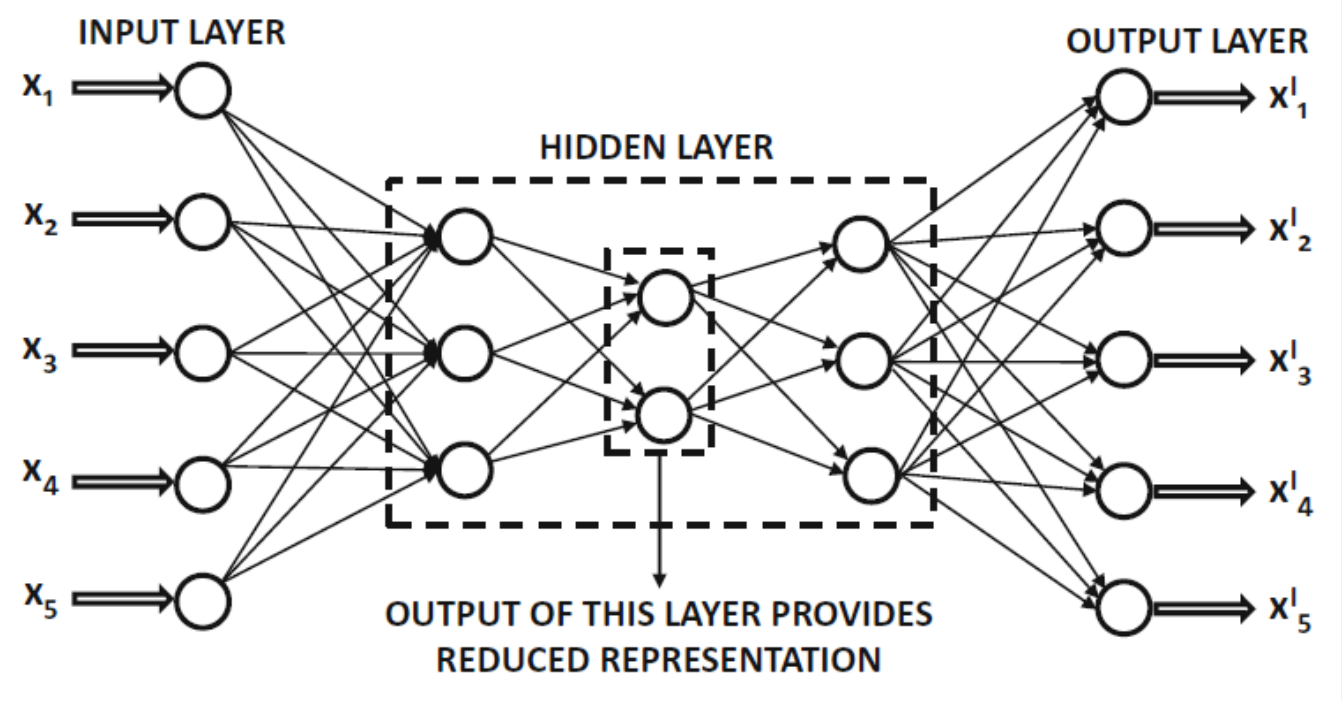


Loss = MSE

AUTOENCODER NO LINEAL

Se incorporan más capas con funciones de activación no lineales.

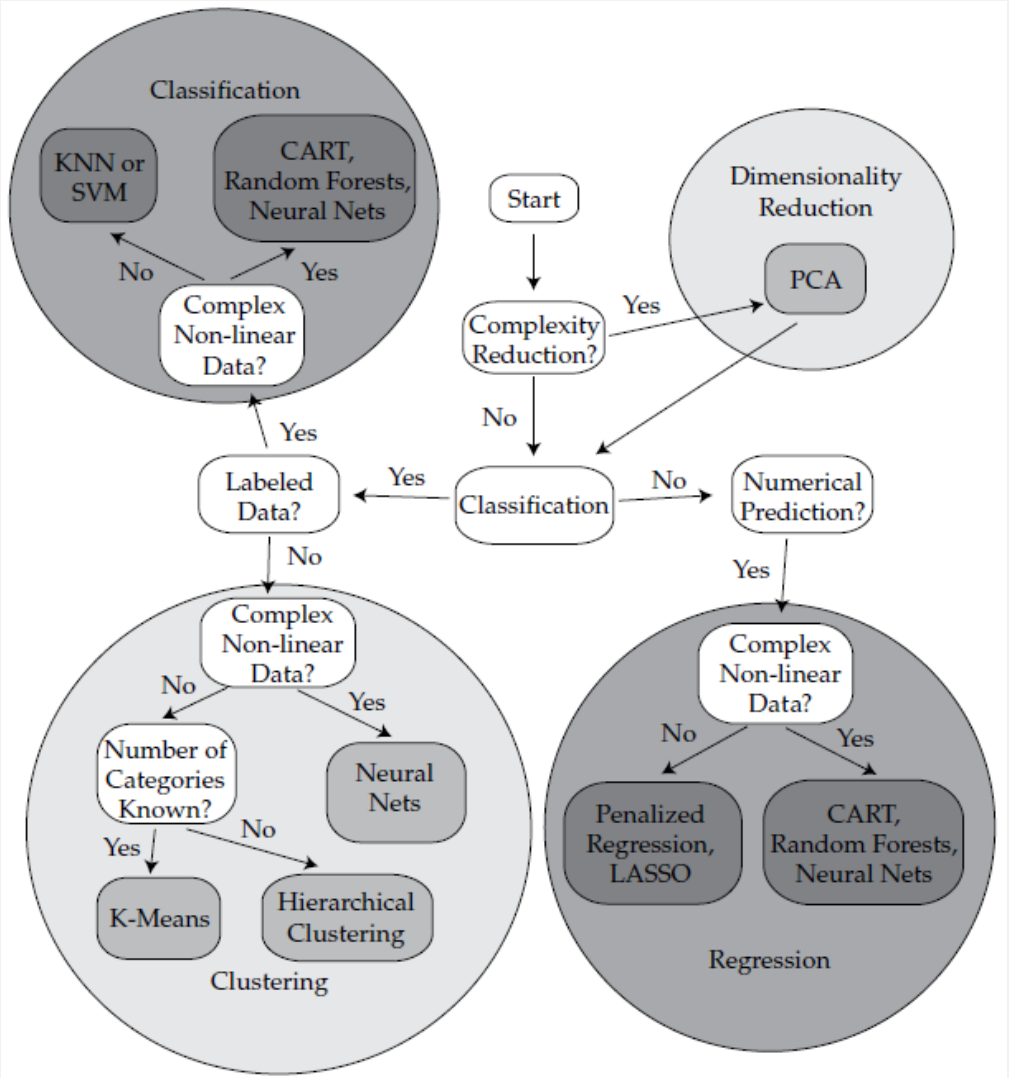
Los weights pueden estar vinculados (**tied**) o no.



Loss = MSE

Comentarios Finales

GUÍA DE SELECCIÓN DE MODELOS DE ML



PARA EXPLORAR

Interpretable Machine Learning

https://christophm.github.io/interpretable-ml-book/

Reinforcement Learning en Economía y Finanzas

https://arxiv.org/pdf/2003.10014

https://www.forbes.com/sites/quora/2018/07/25/what-are-the-latest-works-on-reinforcement-learning-in-the-financial-field/