clusterAI 2020
ciencia de datos en ingeniería industrial
UTN BA
curso I5521

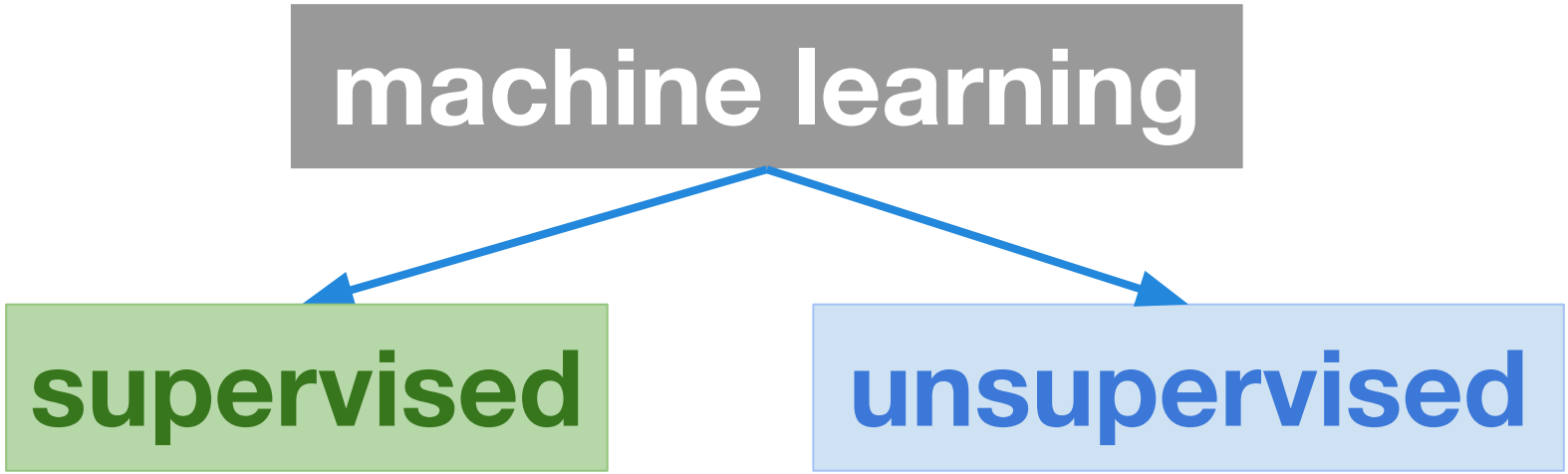
clase_05: Regresion

Docente: Martin Palazzo

agenda clase05: aprendizaje supervisado

- Regresion
- Errors metrics para evaluación del modelo
- Regresión Lineal Multivariada
- Ridge regression
- Lasso regression
- Support Vector Regression
- KNN Regression
- Code

learning approaches



Particularmente en este curso vamos a poner foco en el aprendizaje supervisado y el aprendizaje no-supervisado. Estos dos enfoques suelen ser los más populares y prácticos para la mayoría de los problemas.

Aprendizaje supervisado

$$S = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

$$\mathcal{X} \in \mathbb{R}^d \qquad y \in \mathbb{R} \qquad f(x) = y$$

El enfoque de aprendizaje supervisado se basa en disponer datos ordenados en un dataset **S** en pares de instancias y etiquetas (samples 'x' & labels 'y'). Las instancias son vectores d-dimensionales de variables aleatorias i.i.d. (independientes e idénticamente distribuidos). Las etiquetas se suponen variables dependientes que pueden tomar valores discretos (clases) o continuos a partir de distintos valores de x mediante una función f(x) llamada 'ground truth' o función objetivo tal que f(x) = y. Es decir que f(.) explica la relación entre 'x' (input) e 'y' (output) Como la realidad es compleja generalmente no conocemos la verdadera f(x), por lo que trataremos de aproximarla.

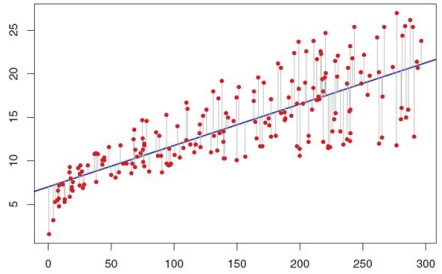
Aprendizaje supervisado

$$f(x) = y \qquad \hat{f}(x) = \hat{y} \qquad L(y, \hat{y})$$

Suponiendo que tanto el dataset de sample-features y las etiquetas están disponibles s=(x,y) vamos a **aprender** una función f'(x) que explique lo mejor posible la relación (x,y). Es decir aquella que aprenderemos una función que tomando como input las variables aleatorias "x" genere un output y' lo más similar a las etiquetas "y" dadas. Para poder medir cuán cerca están las etiquetas generadas por la función aprendida f'(x) utilizaremos una función L(y,y') de Costo o Pérdida (Loss function) que tomará valores altos cuando y sea muy distinto de y' . Por el contrario cuando y sea muy parecido a y' la función de costo tomará valores bajos. Por esta razón buscamos **minimizar** la función de costo.

Métodos de aprendizaje supervisado

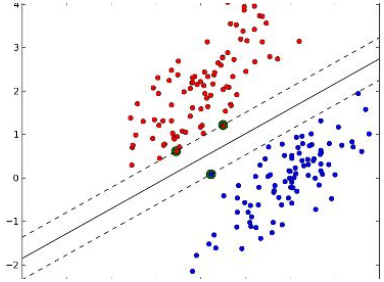
regression



Y es continua

$$y \subseteq \mathbb{R}$$

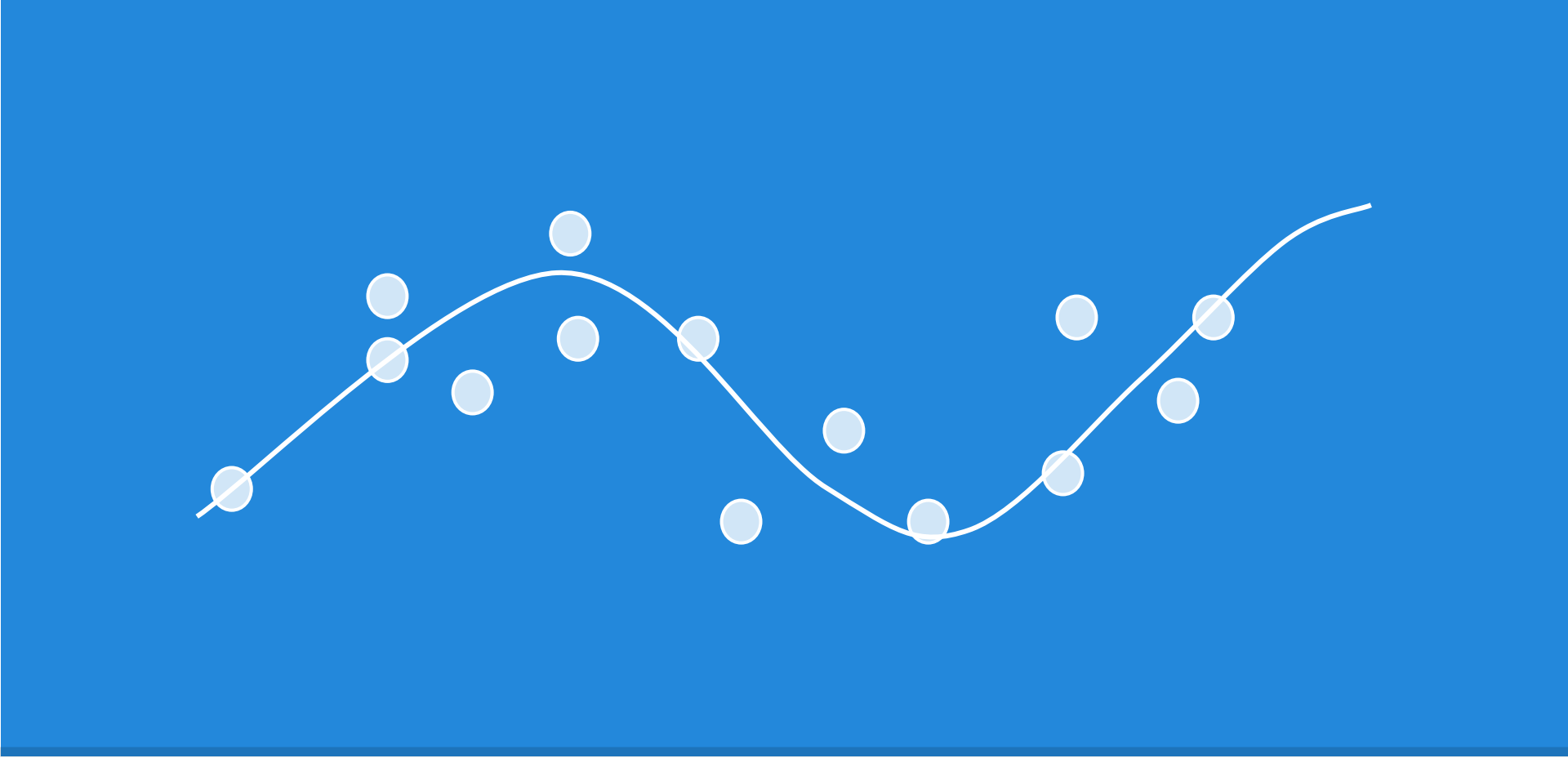
classification



Y es categorica

$$y \in \{-1, 1\}$$

Existen dos enfoques importantes en el aprendizaje supervisado: clasificación y regresión. Cuando las etiquetas toman valores categóricos hablamos de clasificación. Cuando las etiquetas toman valores continuos hablamos de regresión.



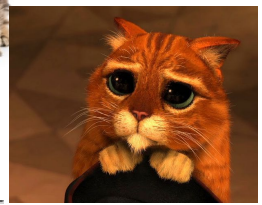
Aprendizaje supervisado: regresión

supervised learning: regression

pic01 X_1

pic02 X_2

pic03 X_3



pic04 X_4

Altura = 1.1 y_1

Altura = 1.2 y_2

Altura = 0.4 y_3

Altura = 0.5 y_4

Data type: images
Features: Pixels

Cada muestra está asociada a una etiqueta continua aplicada por un humano.

supervised learning: regression

pic01 X_1

pic02 X_2

pic03 X_3

Datos de entrenamiento

Altura = 1.1 y_1

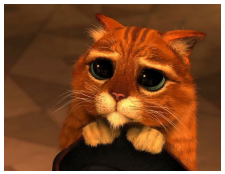
Altura = 1.2 y_2

Altura = 0.3 y_3

Data type: images
Features: Pixels

Entrenamiento de modelo supervisado de regresión

Predicción



Muestra independiente de test (prueba)

pic04 X_4

Altura = ??? y_4

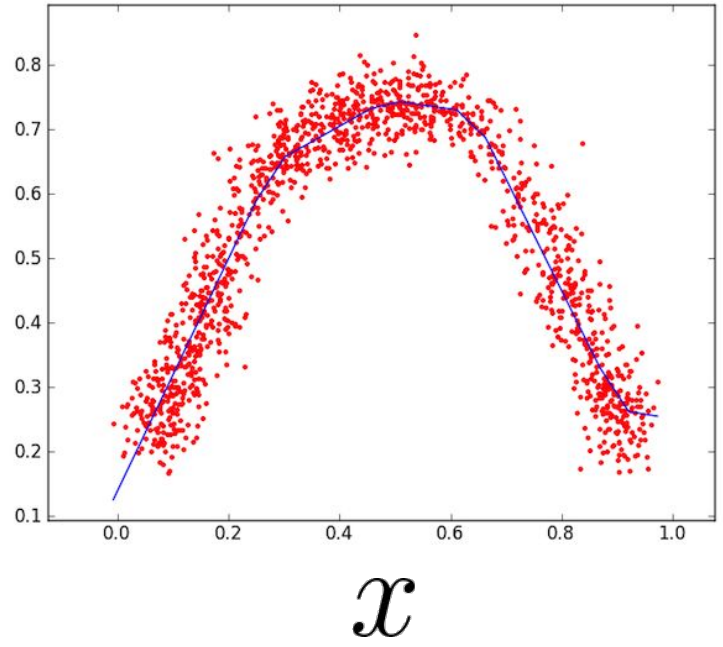
Cada muestra está asociada a una etiqueta continua. No hay clases.

Funciones regresoras

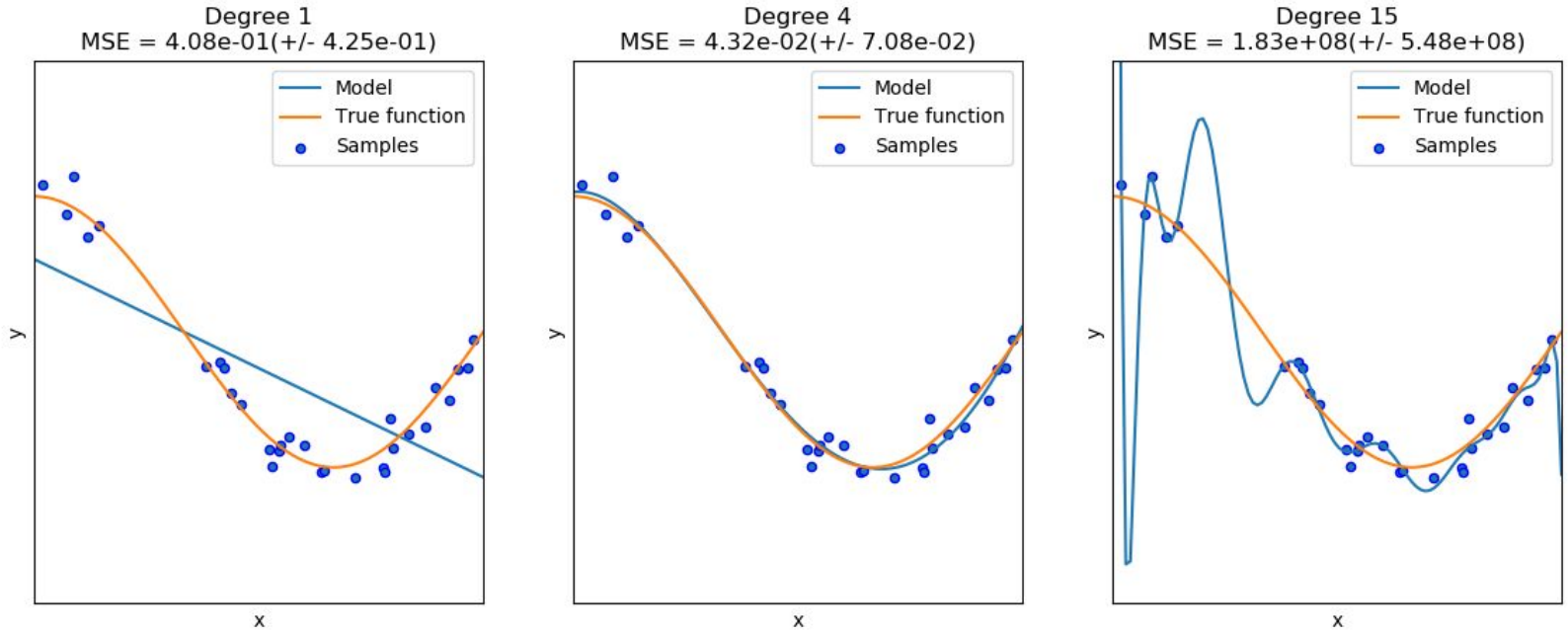
- Es aquella función F(x) que a partir de ciertas “features”/variables de una muestra, estima un valor continuo (real) de la variable dependiente “Y” de esa misma muestra.

- La función regresora se ajustará a los datos de manera tal de minimizar el error entre la predicción “**y’**” y el valor de la etiqueta “**y**”.

$$\hat{f}_w(x)$$



Underfitting vs Overfitting



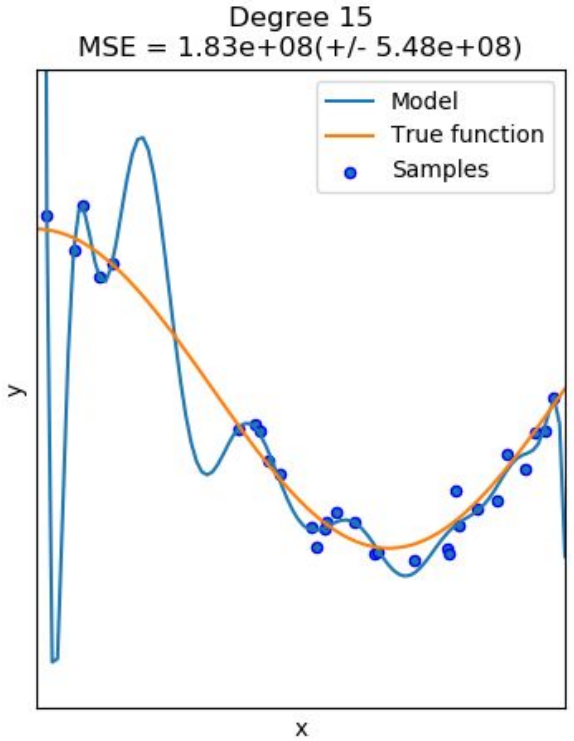
Dado una muestra de datos y suponiendo que conocemos la función que los genera (estocásticamente) ¿cuál modelo es mejor? ¿uno complejo o uno sencillo?

Overfitting vs Underfitting. *Scikit Learn

Overfitting

Un modelo de aprendizaje supervisado sea regresión o clasificación puede tener dos problemas: overfitting (sobre-ajuste) o underfitting (sub-ajuste).

Overfitting: ajustarse demasiado a los datos de entrenamiento para reducir el error de entrenamiento aún cuando suponemos que estos datos no son 100% representativos de la población de donde fueron obtenidos. Es decir, es imposible tener suficiente cantidad de datos como para asegurar que minimizando el error en entrenamiento minimizamos el error de predicción para futuras muestras no vistas. Una forma de observar el overfitting es cuando en train tenemos poco error y en muestras de test independientes tenemos alto error. Sin embargo existen casos donde el error en train y test puede ser muy bajo y aún asi realizar overfitting. **El overfitting suele estar asociado a modelos más complejos que se ajustan muy bien a los datos de train aunque generalizan mal para datos futuros.**

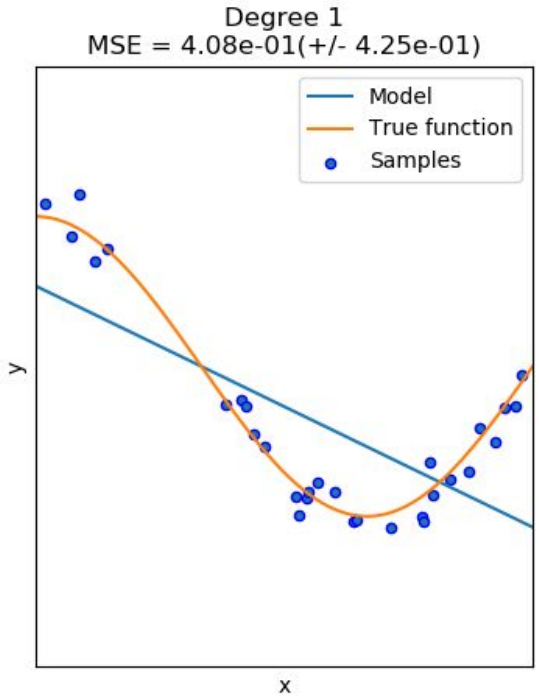


Overfitting vs Underfitting. *Scikit Learn

Underfitting

Un modelo de aprendizaje supervisado sea regresión o clasificación puede tener dos problemas: overfitting (sobre-ajuste) o underfitting (sub-ajuste).

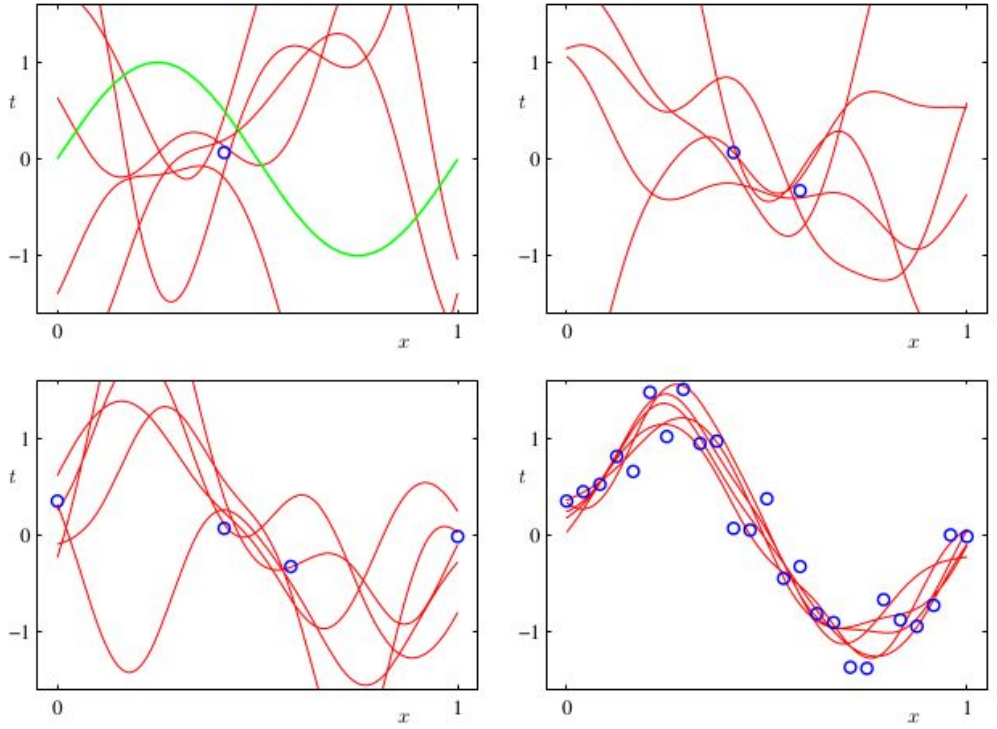
Underfitting: ajustarse poco a los datos de entrenamiento para evitar el sobre-ajuste (overfitting) puede generar un error (Bias) que podría haber sido disminuido y no se hizo. Sub-ajustar un modelo a los datos generará menos variación en resultados futuros aunque un resultado sub-óptimo. **El underfitting suele estar asociado a modelos muy sencillos que no logran ajustarse a la complejidad de los datos (ej. Modelos lineales).**



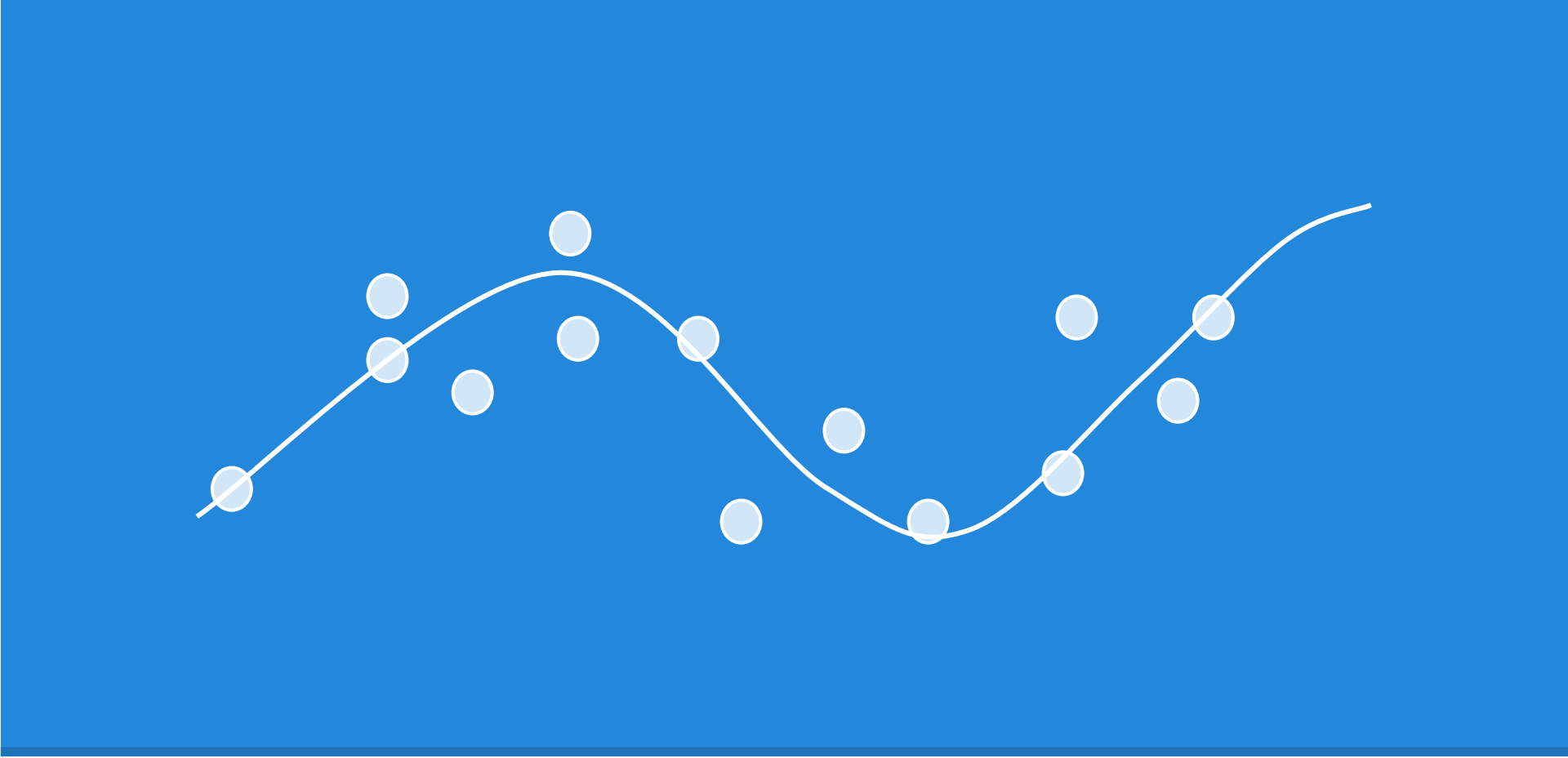
Overfitting vs Underfitting. *Scikit Learn

Regression functions

- La cantidad “**n**” de muestras (samples), determinará el ajuste que realice la función (recordar la curva de aprendizaje).
- Más muestras de entrenamiento (si son representativas de la distribución poblacional) implica comprender mejor la verdadera distribución de mis datos y lograr predicciones en datos futuros con menor error.



*Bishop, Pattern Recognition

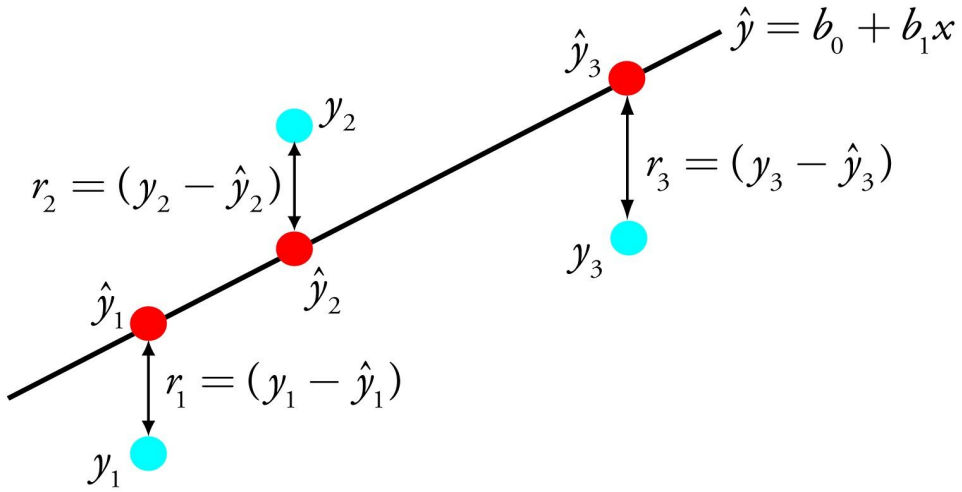


Regresión: medidas de desempeño/performance

Medidas de Error en regresión: como se cuan bueno es mi modelo?

$$e_i = y_i - \hat{y}_i$$

$$\text{RSS} = e_1^2 + e_2^2 + \cdots + e_n^2$$



Cada dif. Entre predicción y valor real se denomina residuo (e). Valores más chicos son mejores ya que aproximamos mejor. La sumatoria de los residuos al cuadrado se llama Suma de Residuos al Cuadrado (RSS = Residual Sum Squares)

R^{2}

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}}$$

$$\text{TSS} = \sum_{i=0}^{i=n} (y_i - \bar{y})^2 \qquad \text{RSS} = e_1^2 + e_2^2 + \cdots + e_n^2$$

El TSS (Total Sum Squares) mide la varianza total de las etiquetas “**y**”. El R2 explica la proporción de la varianza de “**y**” que explica el modelo de regresión. El R2 toma valores entre 0 y 1 y es independiente de la escala de “**y**”.

Medidas de Error en regresión: como se cuan bueno es mi modelo?

$$MAE = \frac{|\sum (\hat{y}_t - y_t)|}{n}$$

$$MSE = \frac{\sum (\hat{y}_t - y_t)^2}{n}$$

$$RMSE = \sqrt{\frac{\sum (\hat{y}_t - y_t)^2}{n}}$$

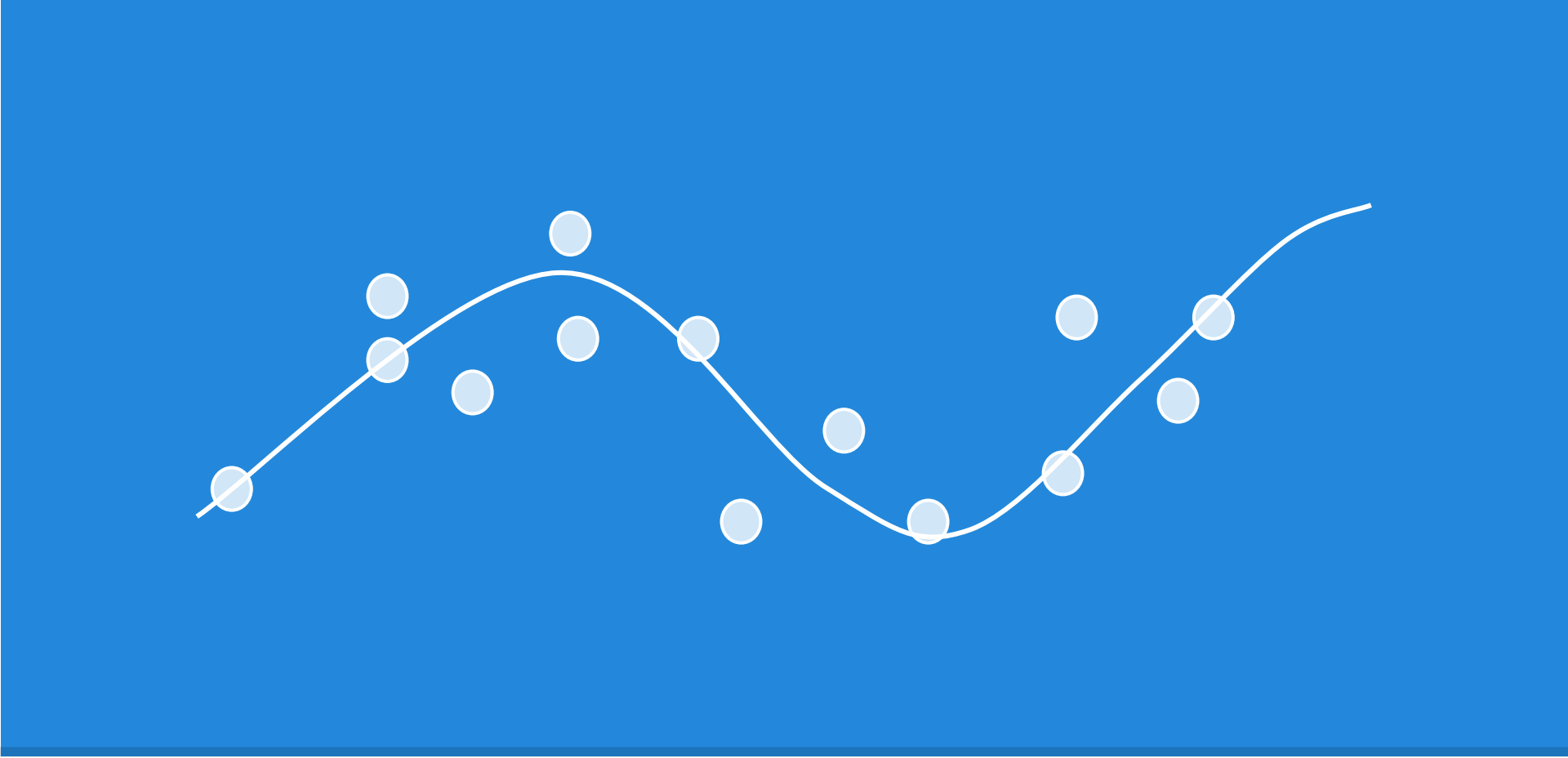
Otras medidas de error en regresión:

1. MSE: Mean Squared Error (error cuadrático medio)
2. RMSE: Root Mean Squared Error (Raiz cuadrada del error cuadrático medio)
3. MAE: Mean Average Error (Media del error)

Por otro lado se observa que:

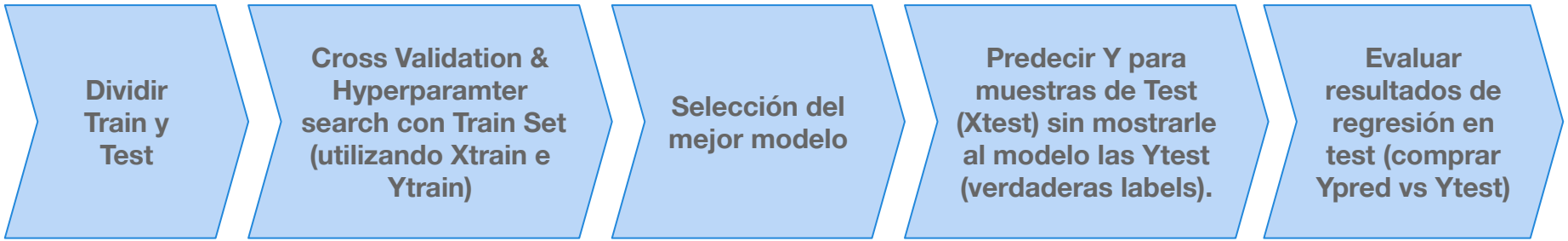
- MSE es sensible a predicciones **muy** malas (residuos grandes). Puede ser problemático en datos ruidosos.
- El RMSE y MAE computa el error en la misma magnitud de la variable dependiente.
- El MAE es mejor cuando no queremos penalizar fuerte errores grandes (ej. outliers).

Queremos encontrar una f(x) que minimice el error E.



Regresión: pipeline de entrenamiento

Pipeline: Train, Validate, Test Model

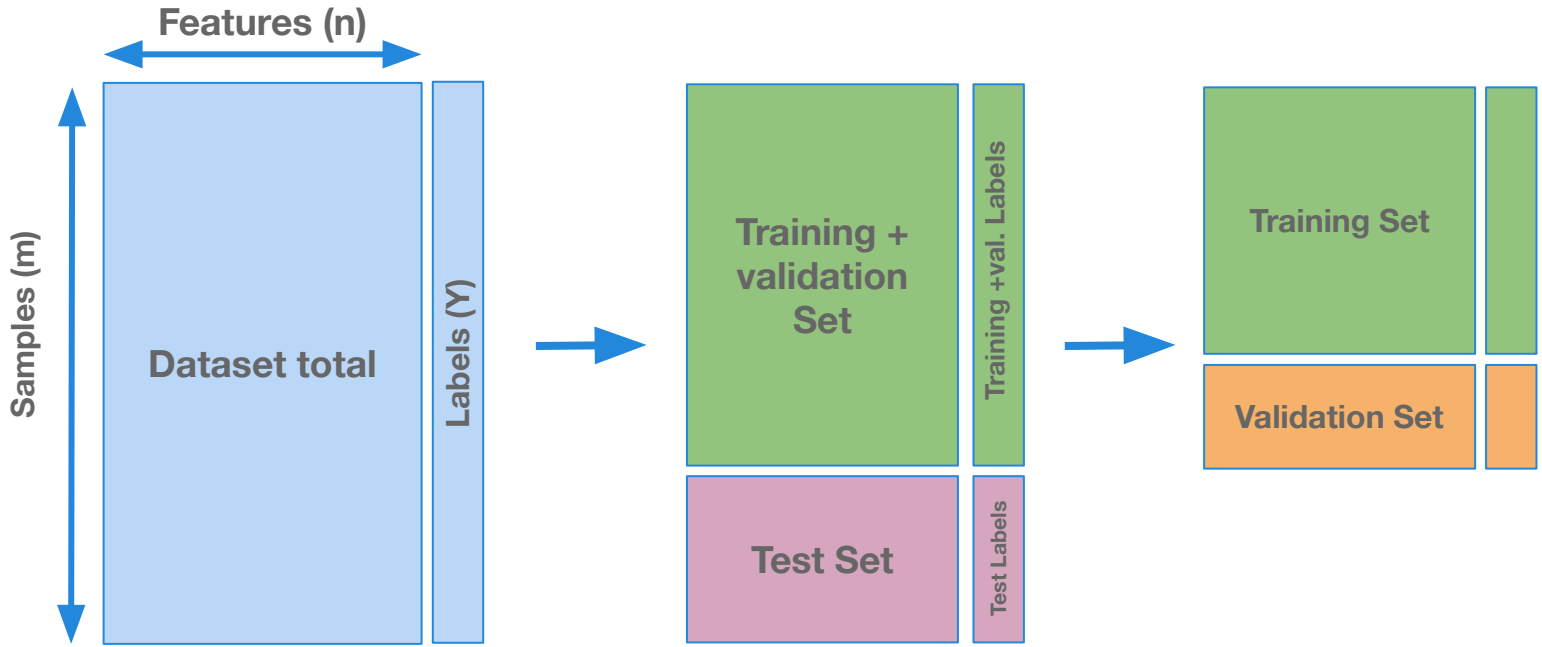


Hiper-parámetros

$$f_{w,\lambda}(x) \quad \text{funcion de decision}$$
$$w \quad \text{parametros}$$
$$\lambda \quad \text{hiper-parametros}$$

Los modelos están caracterizados por parámetros que son aprendidos durante el entrenamiento al ser expuestos a los datos. Adicionalmente los clasificadores tienen hiper-parámetros que definen la familia de funciones que se pueden aprender. Por ejemplo, un hiper-parámetro podría ser el grado de una función polinomial. Los hiper-parámetros no son aprendidos por un algoritmo, son prefijados por el usuario. Los hiperparametros son útiles para poder determinar la complejidad y flexibilidad del clasificador. Por medio de una técnica llamada validación cruzada (cross validation) determinaremos cual la configuración del hiper parámetro que minimiza el error de clasificación.

Train, Validation, Test sets.



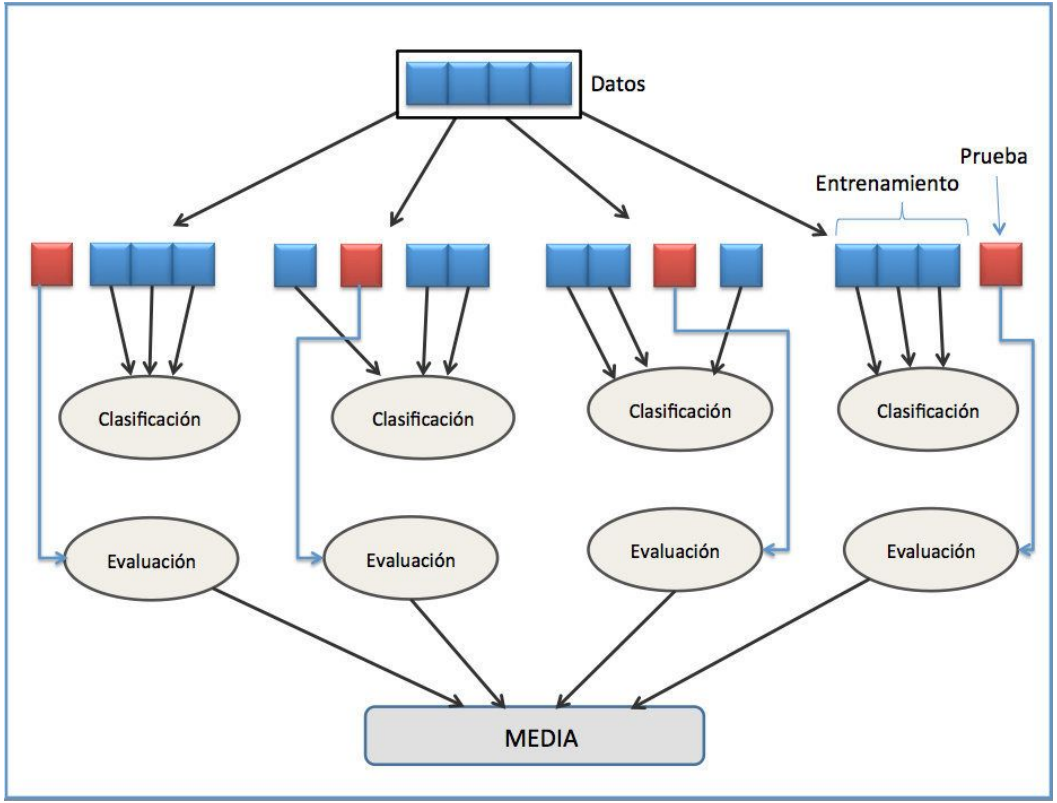
La función de regresión aprenderá utilizando el train set (samples + labels). Luego estimará un valor Ypred para las muestras de test (sin mirar las etiquetas de test). Finalmente se medirá el error de regresión en test.

Cross - Validation en training set

"**Cross validation**" (CV) se realiza con las muestras de entrenamiento. Consiste en dividir nuestro training set en **K folds** (K porciones) e iterar K veces.

En cada iteración, una porción se utiliza como validación independiente y el resto como train. En cada iteración se entrena un modelo con train y se evaluará el resultado de regresión con validación. Luego se realizará un promedio de la exactitud de regresión de las k iteraciones.

Cross validation sirve para poder estimar el error estadísticamente. Además si existen varios hiperparametros a cada uno se estima su error por cross validation y se preserva el hiper parámetro que menor error promedio de cross validation genere.



Modelos de Regresión

Linear Regression, Ridge Regression, Lasso Regression, Support Vector Regression y KNN regression.

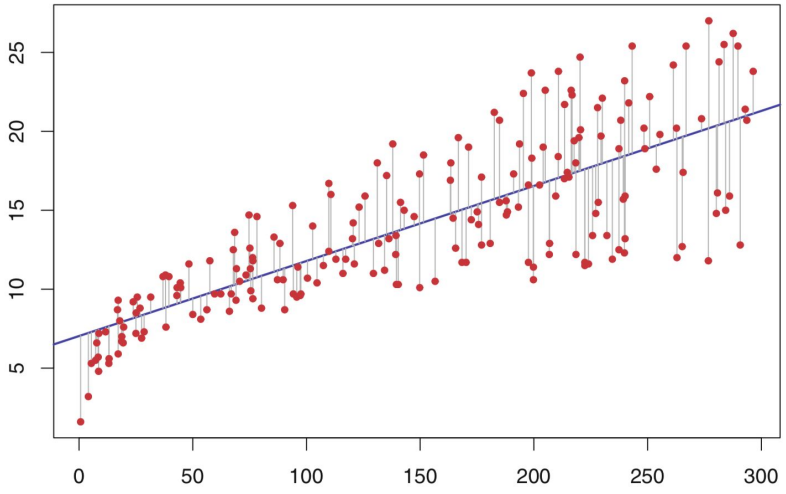
Regresión lineal

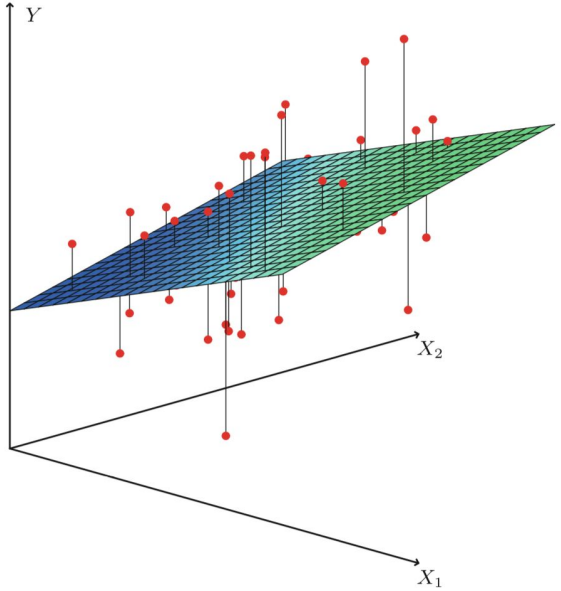
$$f(x) = w^T x = y$$

Función de regresión lineal

Existen muchos tipos de funciones de regresión. La familia de funciones más conocida es la de las funciones lineales. Estas funciones son hiper-planos caracterizados por parámetros **w** (vector w=[w1….wd]) que determinarán los valores que tomará la variable dependiente "**y**" partiendo de los vectores "**x**" que viven en el hiper-espacio de dimensión "**d**".

Regresión Lineal





Por medio de una función lineal caracterizada por parametros **w** (d = 1 izq., d = 2 der.) se buscará minimizar la suma de los residuos al cuadrado. Esta función lineal generará una superficie de regresión en dimensión **d**.

*imagenes: Introduction to Statistical Learning

Regresion Lineal

$$\hat{y} = f(x, w)$$

$$\hat{y}(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_p x_p$$

Es una función lineal que se construye calculando parámetros “**w**” asociados a cada dimensión/feature.

Regresion Lineal

$$\mathrm{RSS}(w) = \sum_{i}^{n} (y_i - f(x_i))^2 = \sum_{i}^{n} \left(y_i - w_0 - \sum_{j}^{d} x_{ij} w_j \right)^2$$

$$\mathrm{RSS}(w) = (\mathbf{y} - \mathbf{X}w)^T (\mathbf{y} - \mathbf{X}w)$$ En forma matricial

Vamos a buscar los parámetros **w** que minimicen RSS por medio del método de Cuadrados Mínimos (Least Squares).

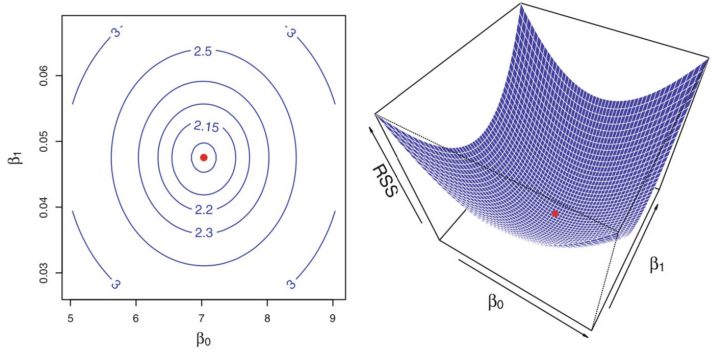
$$\min_{w} \|\mathbf{X}w - \mathbf{y}\|^2$$

Regresión Lineal: Least Squares

Vamos a buscar los parámetros **w** que minimicen RSS por medio del método de Cuadrados Mínimos (Least Squares).

$$\min_{w} \|\mathbf{X}w - \mathbf{y}\|^2$$

Derivando la función de RSS para el caso de regresión lineal podemos obtener una única solución de los valores de los parámetros **w** del modelo (ojo! no son hiper-parámetros) utilizamos Mínimos cuadrados ordinarios (Least Squares) y obtenemos una única solución resolviendo:



*imagenes: Introduction to Statistical Learning

$$\frac{\partial \mathrm{RSS}}{\partial w} = 0$$

$$\hat{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}^T\mathbf{y}$$

Ejercicio

Suponer 3 modelos distintos (a,b,c) ya entrenados y caracterizados con sus respectivos parámetros **w**. Dados los datos de evaluación (X ,Y) calcular el MSE y R2 de cada modelo y seleccionar el mejor.

$$y = w_0 + w_1 x$$

$$\begin{aligned} w_0^{(a)} &= 70 \\ w_1^{(a)} &= -1.5 \end{aligned} \qquad \begin{aligned} w_0^{(b)} &= 50 \\ w_1^{(b)} &= 1 \end{aligned} \qquad \begin{aligned} w_0^{(c)} &= 60 \\ w_1^{(c)} &= -0.5 \end{aligned}$$

$$X = [30, 28, 32, 25, 25, 25, 22, 24, 35, 40]$$
$$Y = [25, 30, 27, 40, 42, 40, 50, 45, 30, 25]$$

Ridge Regression

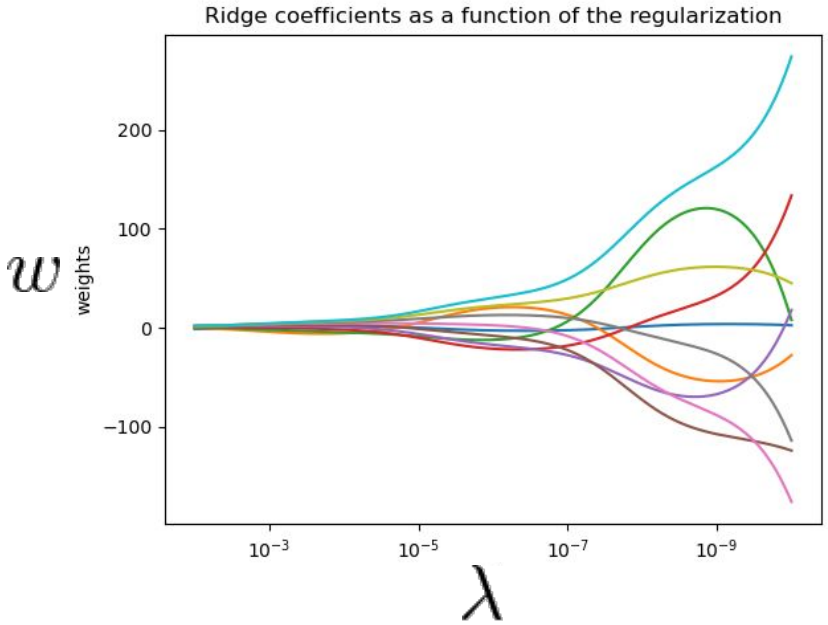
$$\hat{w}^{\text{ridge}} = \underset{w}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - w_0 - \sum_{j=1}^{d} x_{ij} w_j \right)^2 + \lambda \sum_{j=1}^{d} w_j^2 \right\}$$

La regresión Ridge impone una penalización a los parámetros **w** haciendo que estos tiendan a cero en caso de que no sean tan importantes. Se limitará la norma L2 del vector **w**

$$\|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \cdots + |w_d|^2 \right)^{1/2}$$

De manera tal que $\|w\|_2 < t$

Ridge Regression

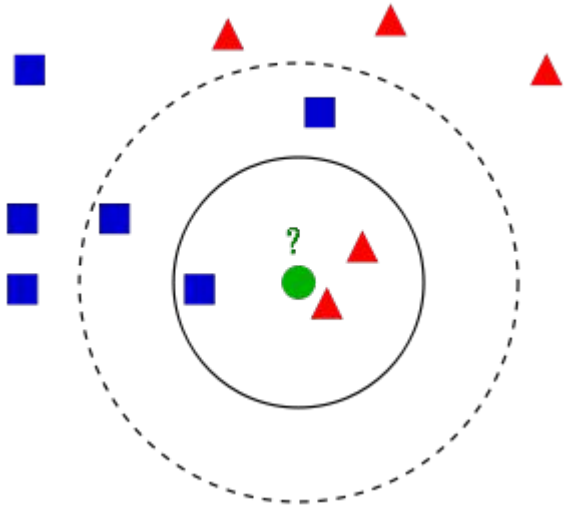


Cuanto mayor sea la penalización entonces más parámetros del vector **w** se aproximarán a cero. Lambda es un hiperparámetro y se obtiene por validación cruzada.

*imagenes: Scikit Learn

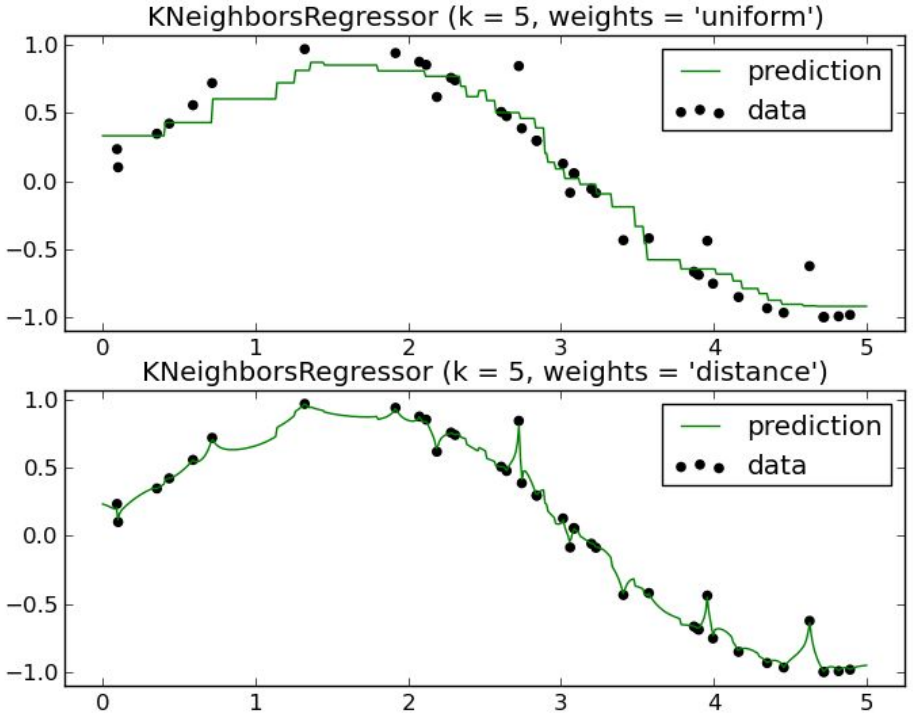
KNN: Regression

- En el entrenamiento se determinan los K vecinos más cercanos por distancia euclídea (distancia par-a-par)
- El Yi a predecir se determina por la interpolación de los Y en los K vecinos (ej por promedio).
- Los pesos **w** indica cómo se interpolara cada K vecino: uniforme (todos por igual) o por distancia.



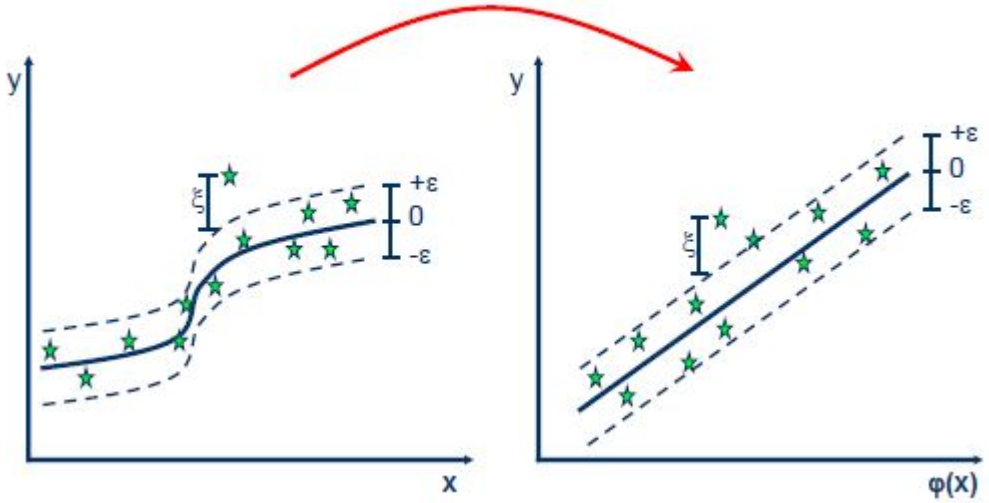
$$d(x_a, x_b) = \sqrt{(x_{a1} - x_{b1})^2 + (x_{a2} - x_{b2})^2 + \ldots + (x_{ap} - x_{bp})^2}$$

KNN: Regression



SVR: Support Vector Regression

- Busca maximizar el margen al igual que SVM, aunque el output es un número real.
- Construye una función Lineal (hiperplano)
- Determina un margen/radio (epsilon) como función de costo y trata de que todas las muestras esten dentro del margen (o “tubo”).
- Con kernels se puede mapear no linearmente mis muestras a otro espacio donde el hiperplano lineal funcione.



$$min \frac{1}{2} \|w\|^2$$

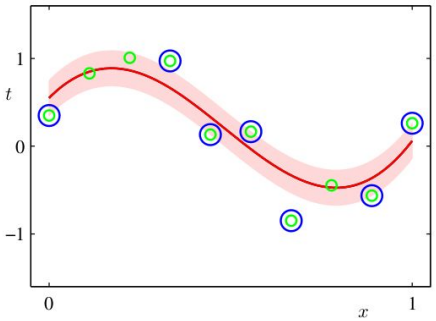
$$y - wx_i - b < \varepsilon$$

$$-y + wx_i + b < \varepsilon$$

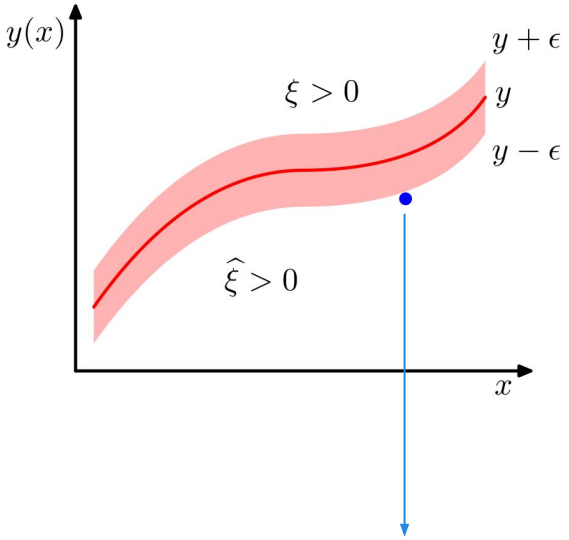
SVR: Support Vector Regression

Las muestras que caigan fuera del “tubo” tendrán “xsi > 0” y las que estén dentro del tubo “Xsi = 0”. El hiper-parámetro es una función **C** que penaliza muestras fuera del “tubo”.

$$C \sum_{n=1}^{N} \xi_n + 1/2 \left\| w \right\|^2$$



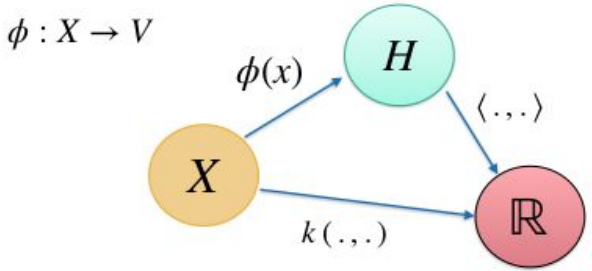
Si el error entre predicción (Ypred) y etiqueta verdadera (Yreal) es menor que epsilon entonces se determina que el error es 0.
Solo algunas muestras definirán el “tubo” de predicción y serán llamadas “support vectors”,



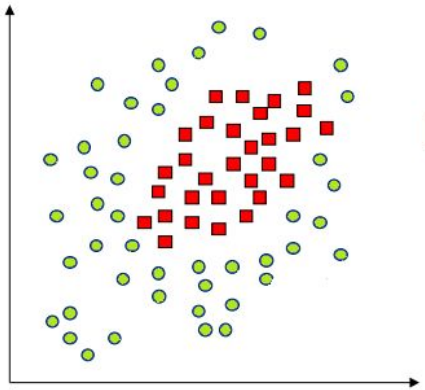
Al igual que en SVM, podemos “permitir/flexibilizar muestras fuera del “tubo” de predicción del modelo.

Kernels in regression: SVR “Kernel Trick”

$$k\left(x_i, x_j\right) = \left\langle \phi(x_i), \phi(x_j) \right\rangle_H$$

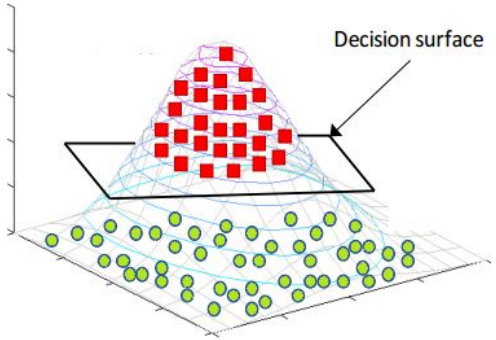


No separables linealmente



kernel

Separables linealmente por un hiperplano.



Los kernels son funciones de similaridad entre muestras. Mapean nuestros datos a una dimensión desconocida (de Hilbert) donde son linealmente separables. Allí en ese nuevo espacio donde son mapeadas las muestras se aplican los productos internos (o similaridad). Cuando usamos SVR, podemos aplicar un kernel para facilitar la regresión, es decir que el hiperplano estará afectado por el kernel de manera **no lineal.**

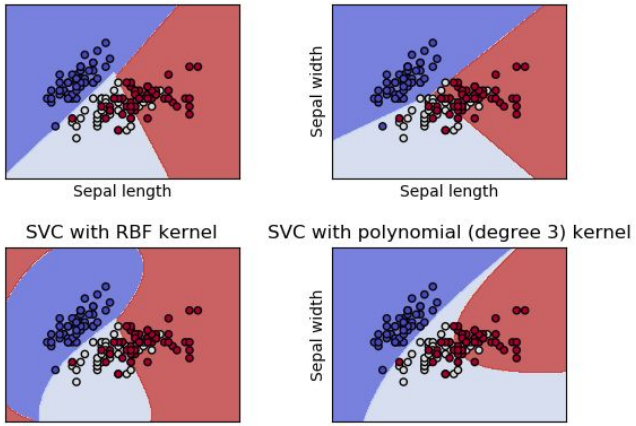
SVM: Hiper Parametros

Kernels más frecuentes: Gaussian, Linar, Polynomial

$$K_{gaussian}(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

$$K_{lin}(x_i, x_j) = \langle x_i, x_j \rangle$$

$$K_{poly}(x_i, x_j) = (\langle x_i, x_j \rangle + R)^d$$



Cada kernel hará que el SVR genere distintos tipos de regla de regresión. Los **kernels gaussianos y polinomiales generarán funciones no lineales mas complejas** y el lineal o el polinomial de bajo grado mas sencillas. ¿cuando usamos un kernel complejo y cuando uno sencillo?

***Kernel Methods for Pattern Analysis, Nello Cristianini**

A CODEAR

scikit learn