

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

clase_03: clasificación

Docente: Martin Palazzo

agenda clase03: aprendizaje supervisado

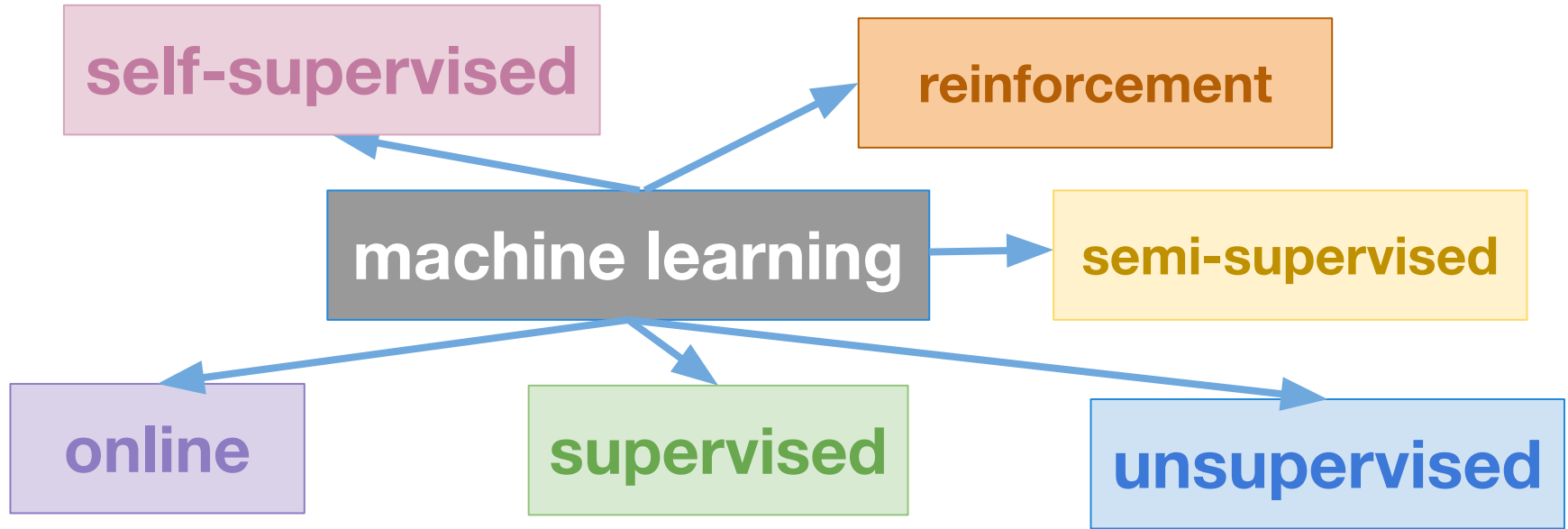
- Clasificación, binary class, multiclass
- Train, validation, test
- Cross validation
- Grid Search
- Confusion Matrix
- Performance metrics (Sens, Spec, ROC)
- Learning curve
- Clasificadores: SVM, KNN, Logistic Regression

AI & Art: Helena Sarin



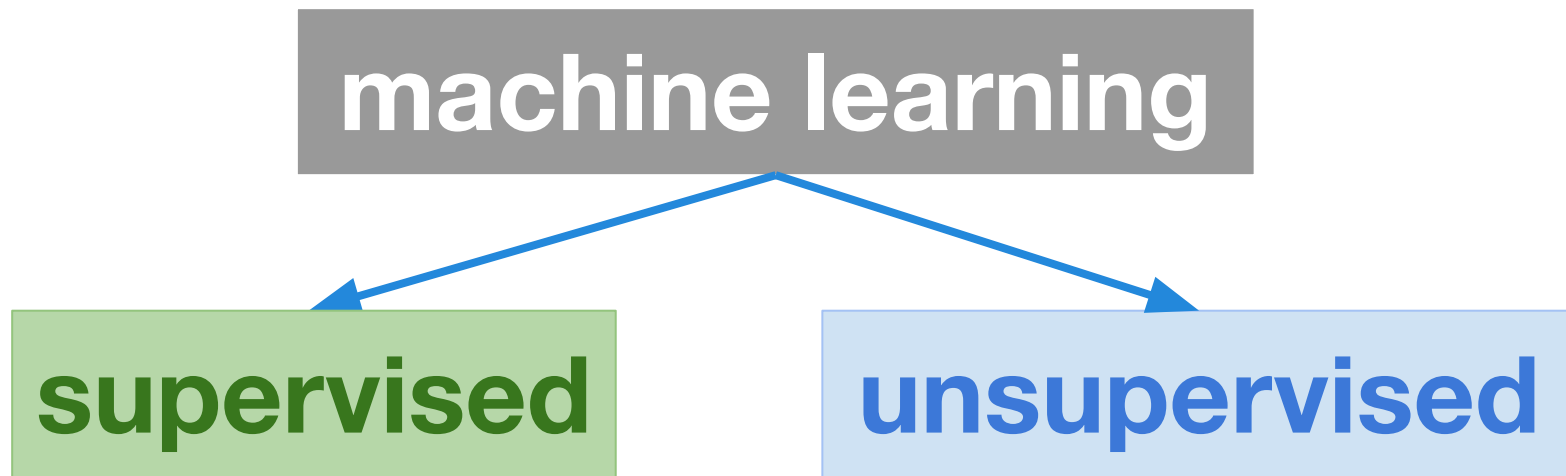
<https://www.neuralbricolage.com/> y <https://twitter.com/glagolista> Modelos entrenados: **CycleGAN**

learning approaches



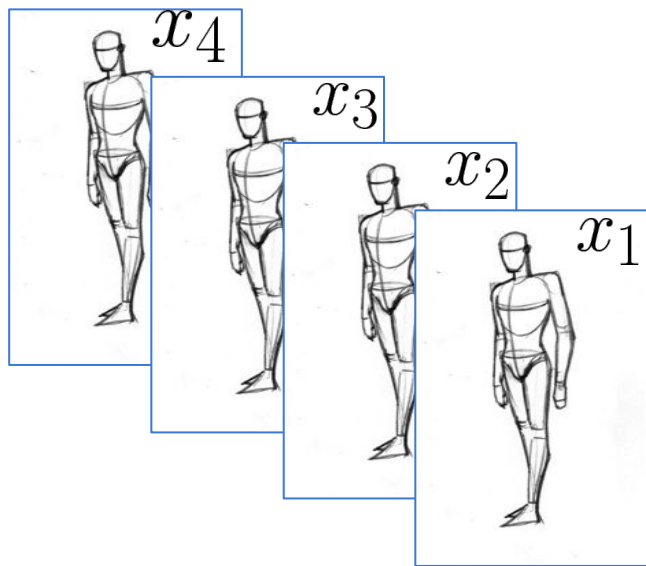
Existen muchos enfoques de aprendizaje automático, cada uno con sus particularidades. Lo que comparten estos enfoques es que todos postulan modelos que aprenden reglas matemáticas y estadísticas gracias a que son expuestos a un conjunto de datos muestreados con el fin de realizar acciones varias. Estos modelos son de distinta complejidad y requieren capacidad de cómputo para ser ejecutados.

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.

Samples and Features



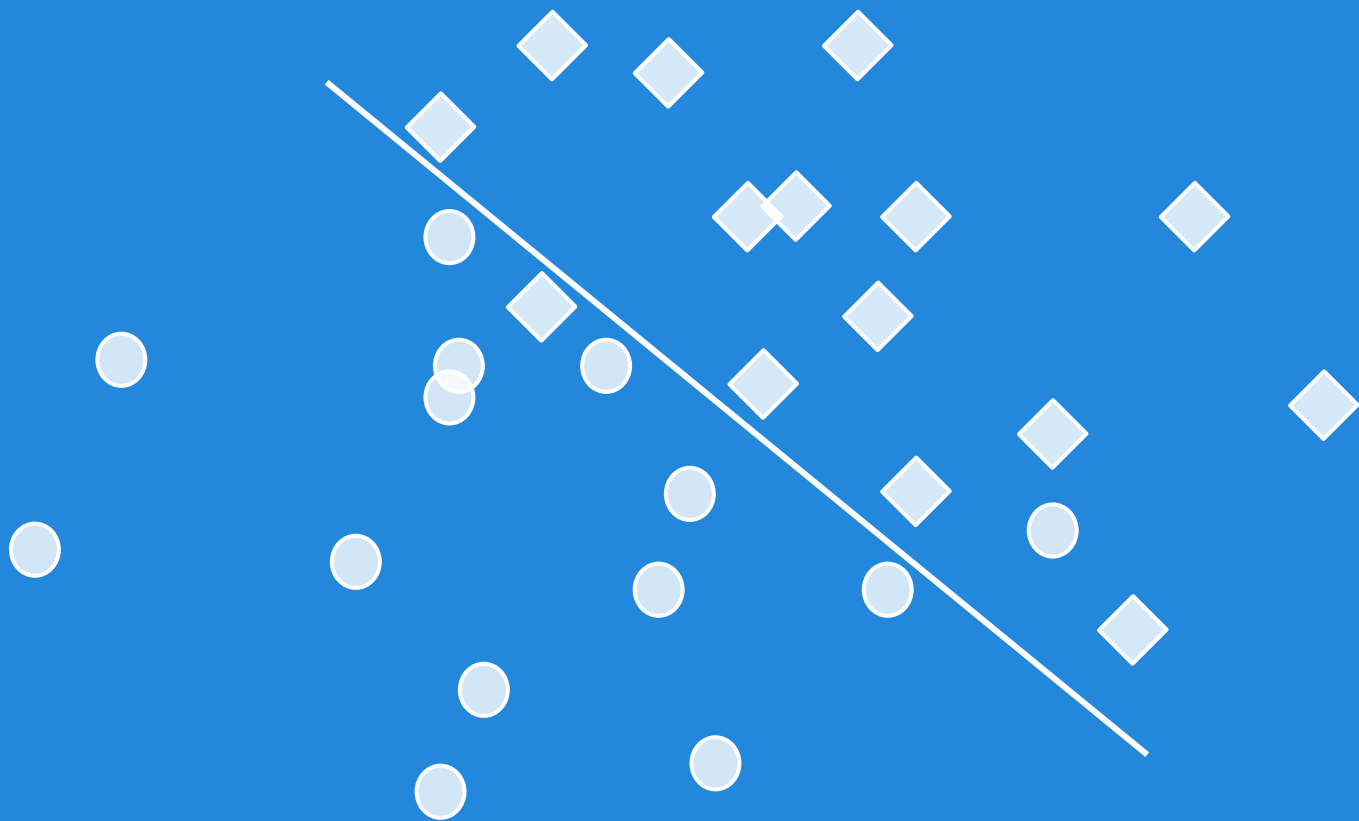
$$X_i = [x_{i1}, x_{i2}, x_{i3}, \dots, x_{id}]$$



$$\mathcal{X} \in \mathbb{R}^d$$

Para que los modelos de machine learning puedan aprender es necesario exponerlos a un conjunto de instancias/samples muestreados de una distribución de probabilidad compleja desconocida. Cada instancia/sample/muestra está caracterizada por un conjunto de features/variables/dimensiones.

Cada sample puede verse como un vector de dimensión d.



Aprendizaje supervisado: clasificación

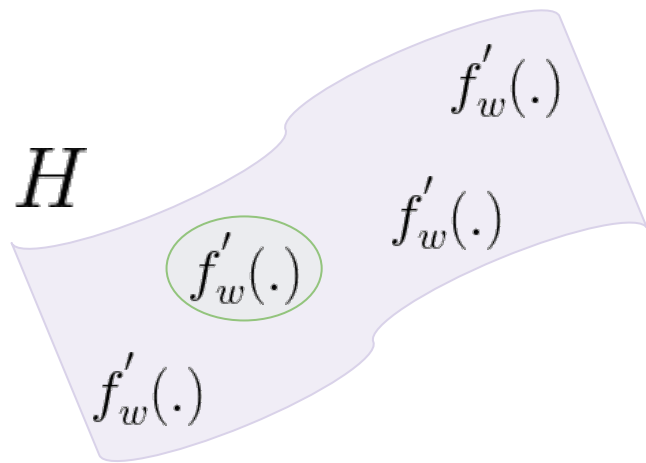
Aprendizaje supervisado

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

$$x \in \mathbb{R}^d \quad y \in \{-1, 1\} \quad f(x) = y$$

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.

Hipotesis



$$H = \left\{ f_w^1(\cdot), f_w^2(\cdot), \dots, \textcircled{f'_w(\cdot)}, \dots \right\}$$

Para aproximarnos a la verdadera $f(x)$ vamos a buscar alguna función $f(\cdot)$ dentro de un espacio de hipótesis que contiene muchas funciones $f(\cdot)$. De todas las funciones disponibles dentro del espacio de hipótesis H vamos a tratar de encontrar alguna que explique lo mejor posible la relación entre el input 'x' y el output 'y'. La función que vayamos a buscar estará caracterizada por **parámetros (theta)** que pueden tomar distintos valores. Entonces existirá una combinación de parámetros que determinen una $f'(\cdot)$ que se aproxime a la verdadera $f(\cdot)$ mas que otras $f'(\cdot)$.

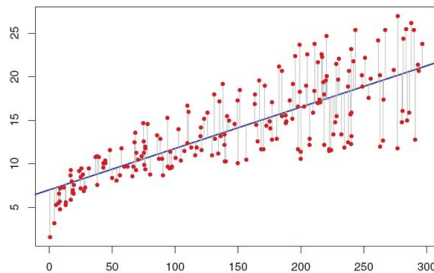
Aprendizaje supervisado

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

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

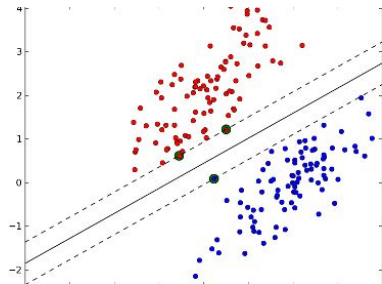
regression



Y es continua

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

classification

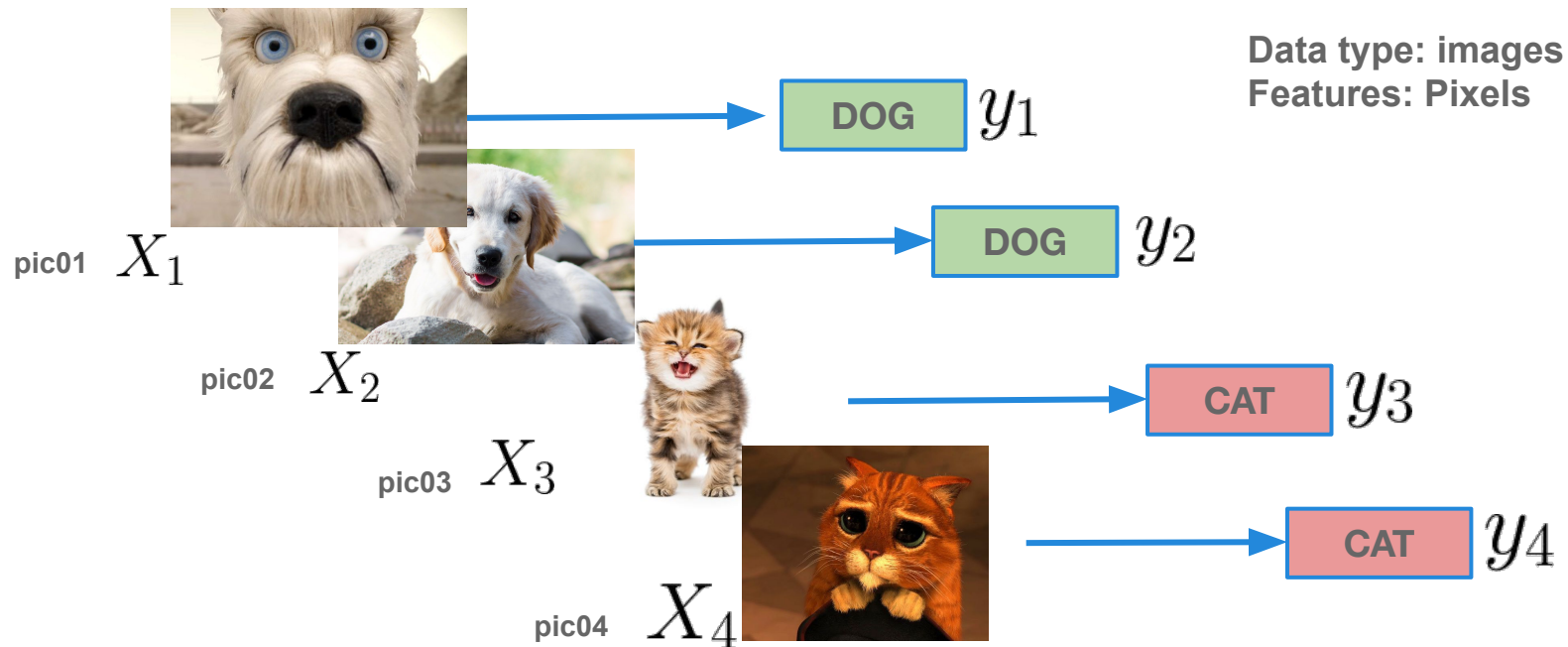


Y es categorica

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

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.

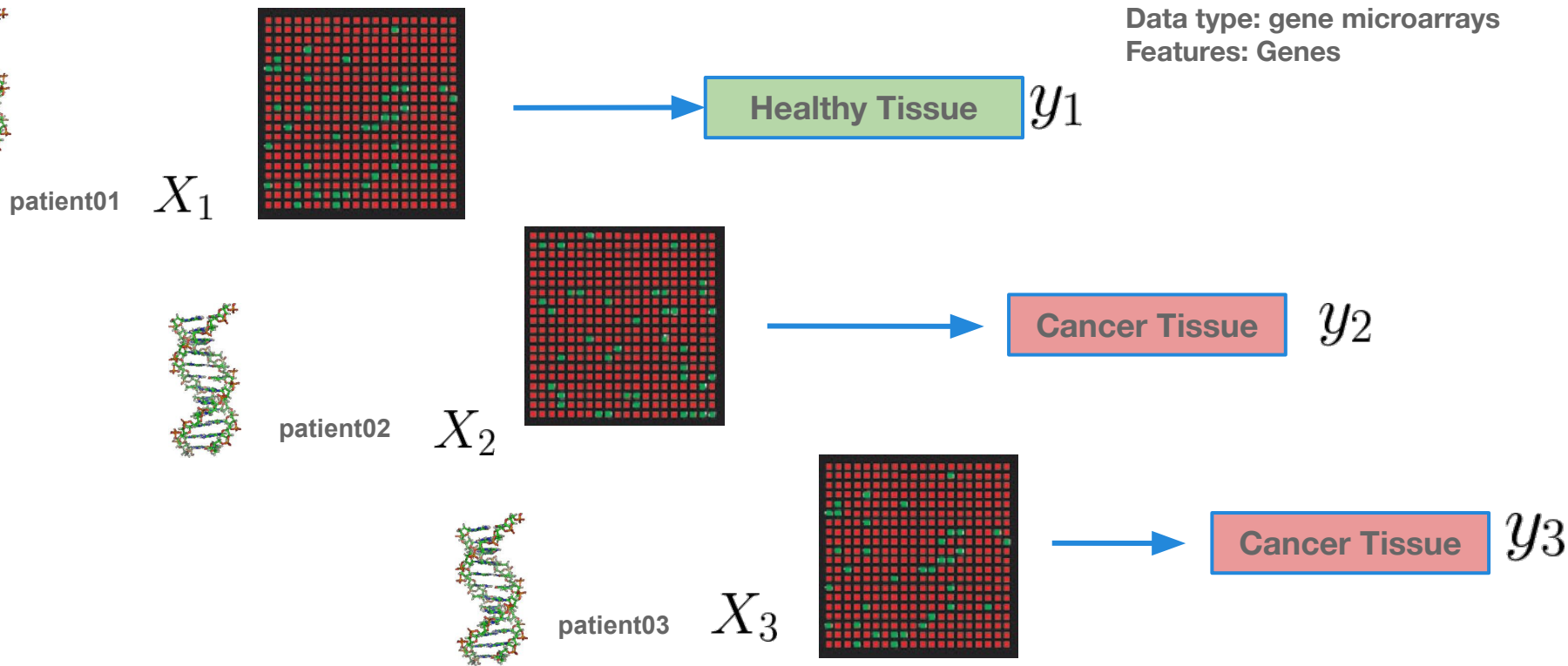
Aprendizaje supervisado: clasificación



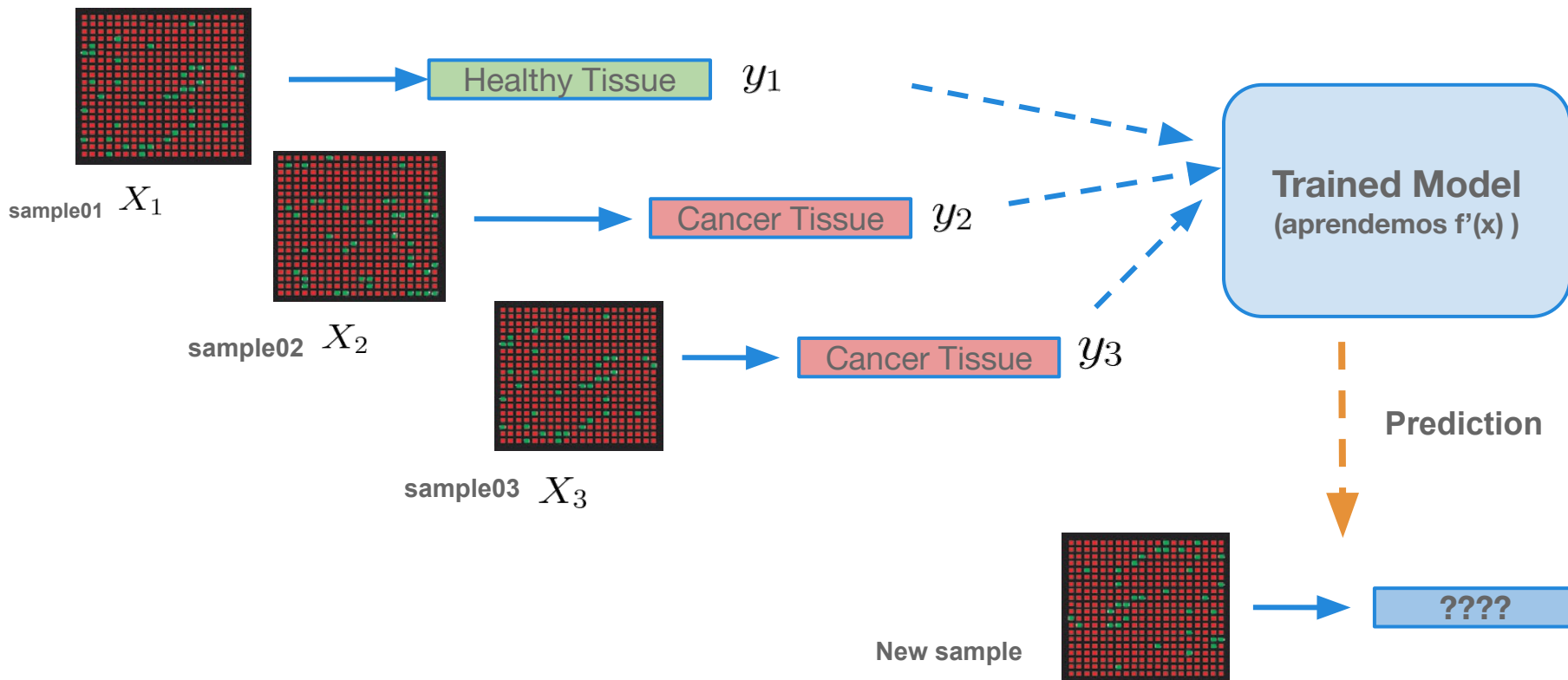
Cada muestra está asociada a una etiqueta categórica (clase) confirmada por un humano.

Aprendizaje supervisado: clasificación

Data type: gene microarrays
Features: Genes

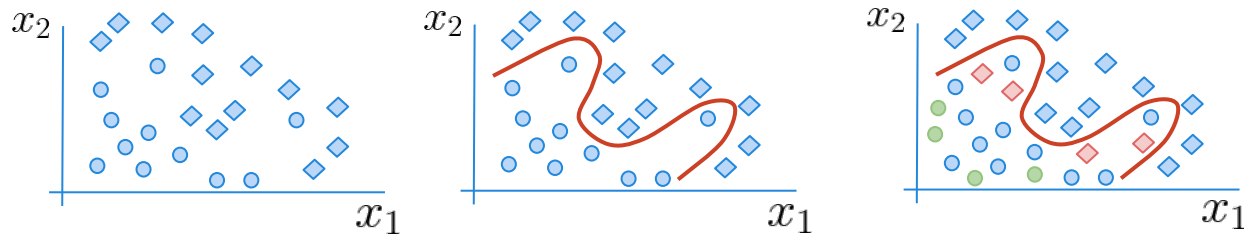


Aprendizaje supervisado: clasificación

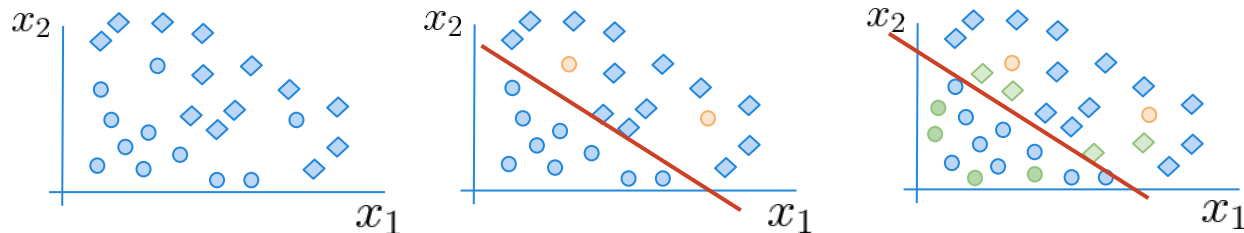


Aprendizaje supervisado: clasificación

Modelo no-lineal



Modelo lineal



Datos de entrenamiento con etiquetas de dos clases.

Funcion de clasificacion: entrenamos un modelo no-lineal y lineal.

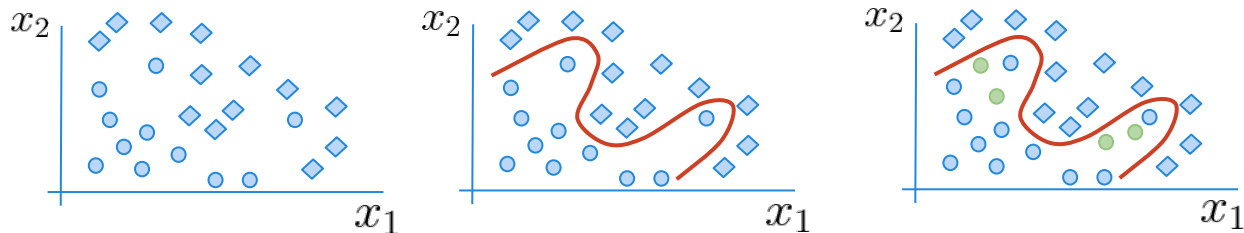
Agregamos 4 muestras nuevas de cada clase y vemos cómo clasifican con el modelo entrenado.

Durante el entrenamiento el modelo estará expuesto a muchas muestras de “entrenamiento” y sus respectivas etiquetas de manera tal de que aprenda la regla $f'(x)$ que **minimiza el error de clasificación**. Simultáneamente esperamos que luego de entrenar la regla/función de decisión aprendida en los datos de entrenamiento funcione “bien” para muestras que nunca vió (test), es decir, que pueda **generalizar** a instancias ‘x’ nunca vistas clasificándolas correctamente.

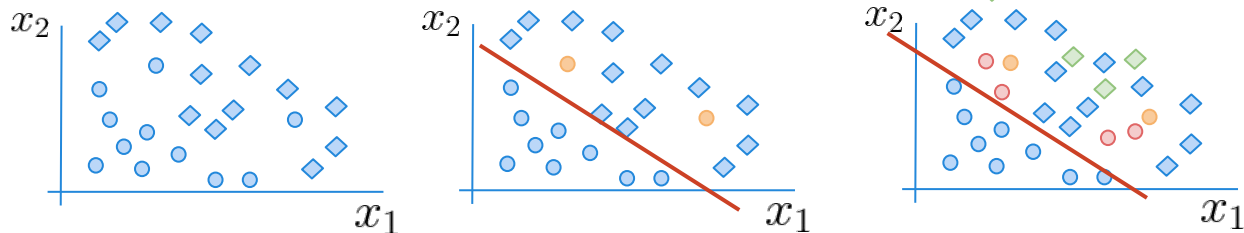
En la figura: cuál modelo tiene menor error en el entrenamiento? cual modelo generaliza mejor? Cual es mas complejo? Cual es mas sencillo?

Aprendizaje supervisado: clasificación

Modelo no-lineal



Modelo lineal



Datos de entrenamiento con etiquetas de dos clases.

Funcion de clasificación: entrenamos un modelo no-lineal y lineal.

Agregamos 4 muestras nuevas de cada clase y vemos cómo clasifican con el modelo entrenado.

Dados los mismos datos de entrenamiento del caso anterior, volvemos a entrenar los dos modelos: lineal y no lineal. Esta vez con otras muestras nuevas (test). Los resultados de clasificación son distintos en este caso. Cual de los dos modelos es mejor? Cual es la principal diferencia entre ambos modelos? Cual de los dos debemos mantener para el análisis?

Frontera de decision lineal

Hiper-plano separador

$$f(x) = w^T x + b = 0$$

Funcion de decision

$$D(x) = \text{sign}[w^T x + b]$$

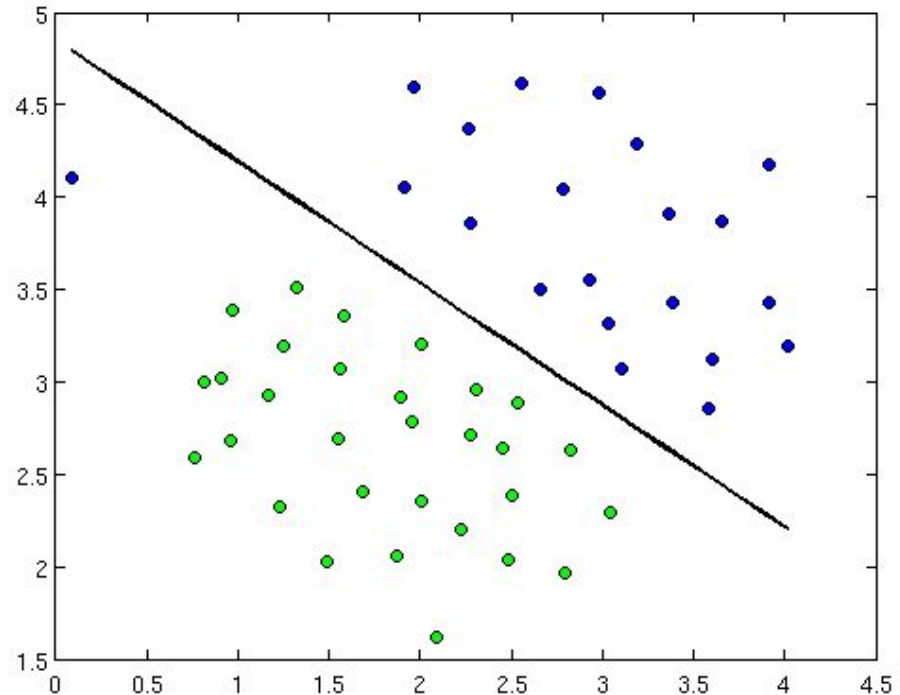
Existen muchos tipos de funciones de decisión. La familia de funciones más conocida es la de las funciones lineales. Estas funciones son hiper-planos caracterizados por parámetros \mathbf{w} (vector $w=[w_1 \dots w_d]$) que determinarán cómo se posiciona la frontera de decisión en el hiper-espacio de dimensión d . En la clasificación binaria la función de decision asignara un valor de $\mathbf{y=1}$ o $\mathbf{y=-1}$ según de que lado del hiper-plano se posicionen las muestras \mathbf{x} .

Funciones de decisión

Una función de decisión toma un vector input X (sample) con “ d ” features, y le asigna una de las K clases, llamada C_k .

- Cuando $C_k = 2 \rightarrow$ binaria
- Cuando $C_k > 2 =$ multiclase

En el ejemplo de la derecha tenemos dos dimensiones y dos clases. La función de decisión es una función lineal que clasifica casi todas las muestras bien menos una sola (dependiendo del lado que se encuentre la muestra respecto al clasificador).

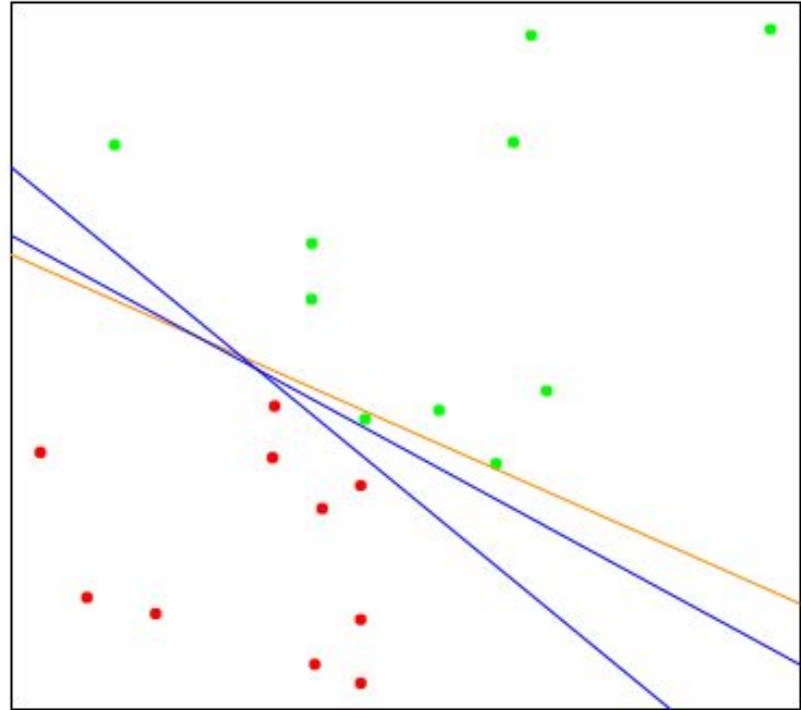


Funciones de Decisión

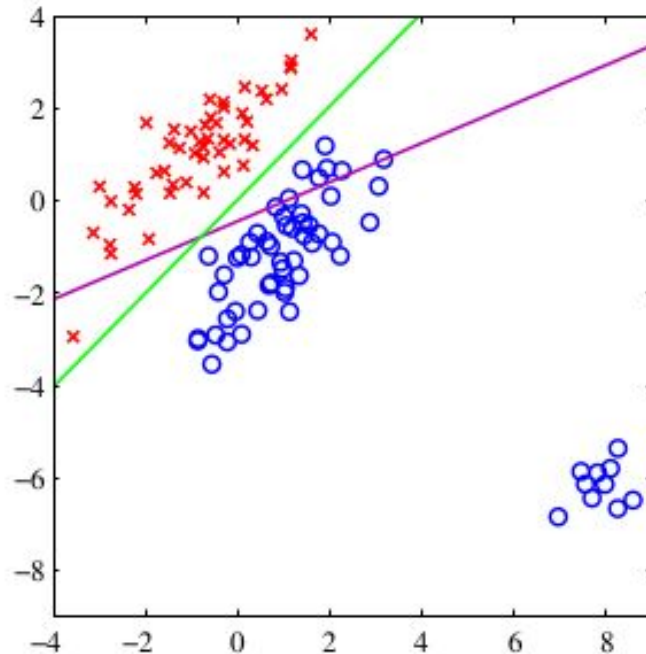
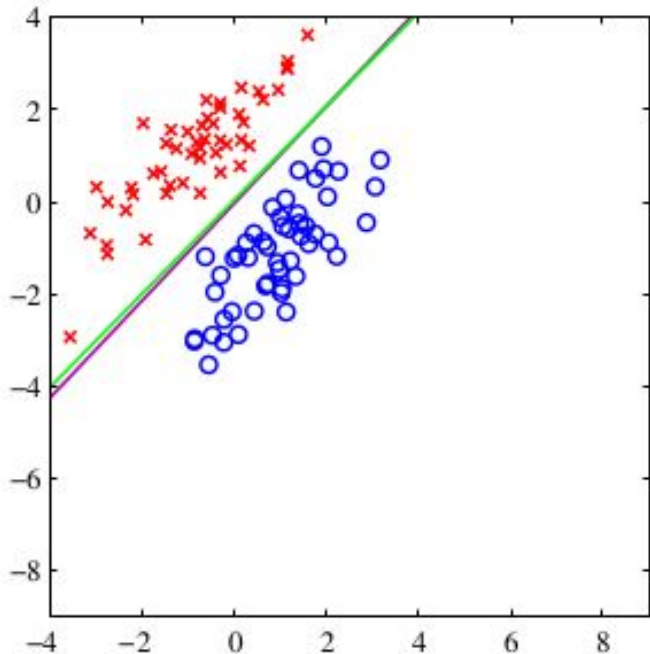
Para un mismo set de datos etiquetados, distintos modelos pueden generar **distintas** funciones de decisión (decision rules).

Algunos modelos generarán funciones de decisión más **sencillas** y otros aprenderán funciones más **complejas**.

La complejidad de la función a aprender dependerá de la complejidad de la **distribución** de las muestras de entrenamiento.



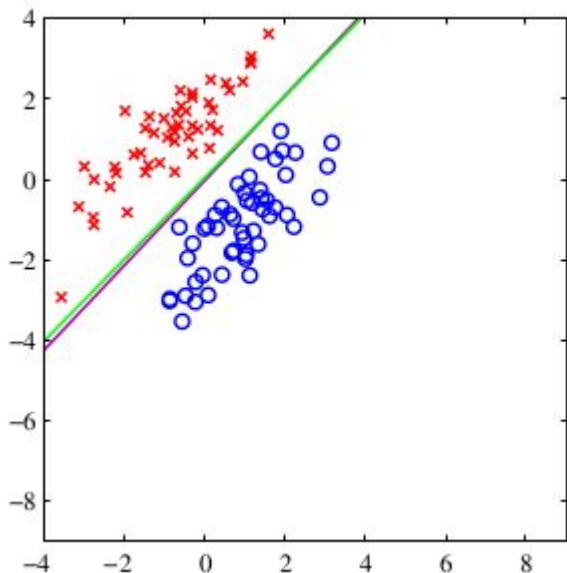
Funciones de Decisión



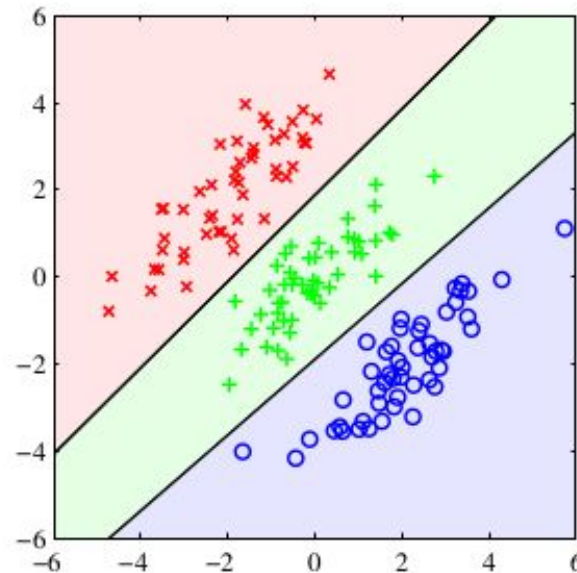
¿cuál clasificador es mejor? **Bishop, Pattern Recognition*

Tipos de clasificación

Clasificación Binaria



Clasificación Multiclase



En clasificación binaria aprenderemos una sola función (clasificador) mientras que en multiclase será k o $k-1$ funciones según el caso.

Variance vs Bias

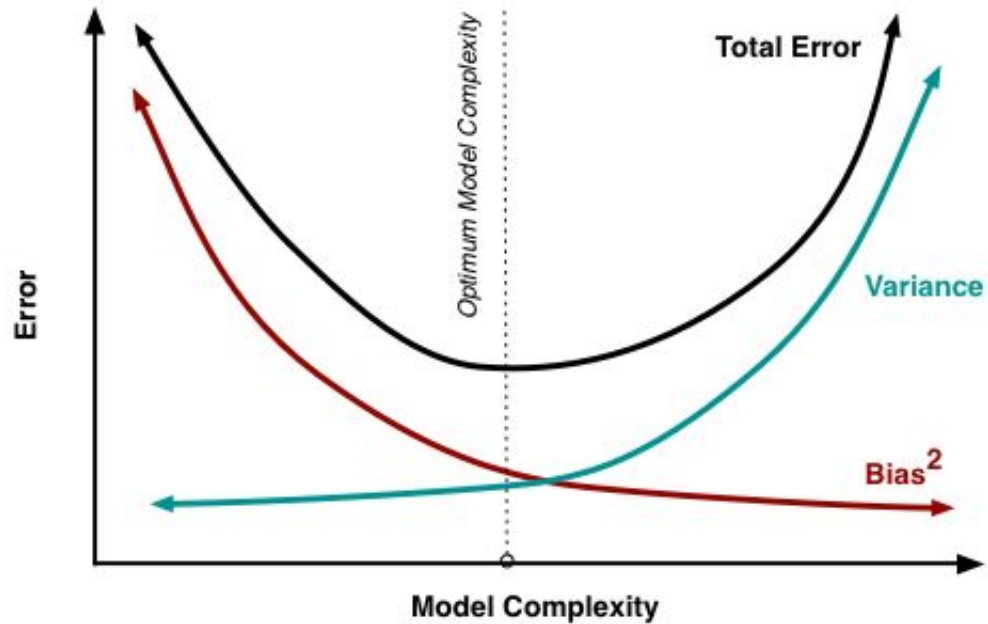
$$\mathcal{L}(y, \hat{y}) = \text{Var}(\hat{f}(x)) + [\text{Bias}(\hat{f}(x))]^2$$

Cuando la etiqueta real y es distinta a la etiqueta asignada por el clasificador \hat{y} la función de pérdida-costo (Loss) \mathcal{L} se incrementará. La causa de que un modelo supervisado tenga error en muestras nuevas se puede causar por su sesgo o por su varianza.

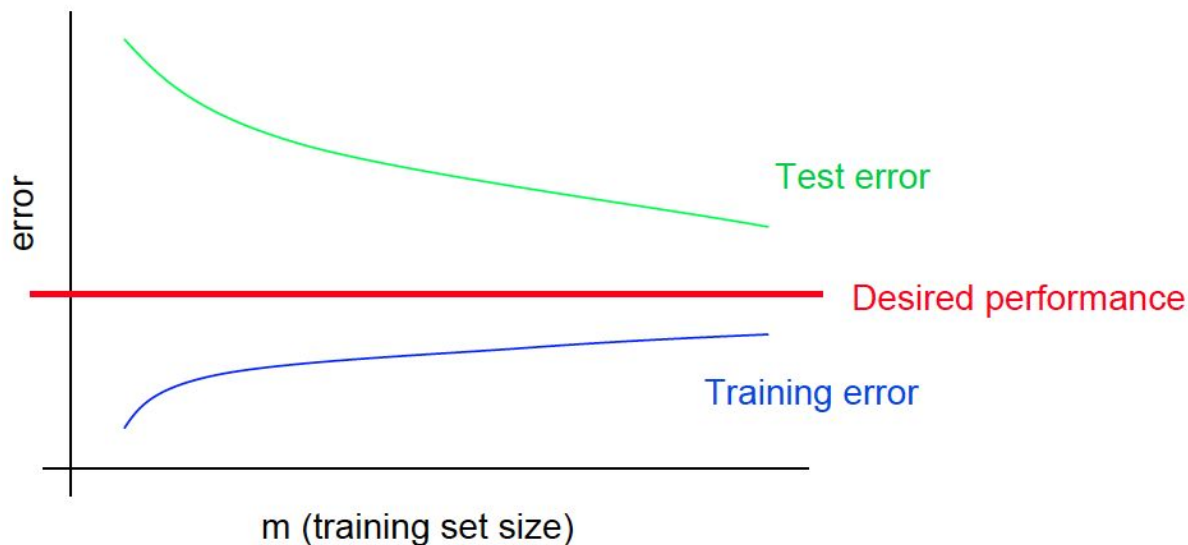
Un modelo tiene alta varianza cuando su predicción varía mucho si lo exponemos frente a muestras distintas provenientes de la misma población. En general la varianza se observa en modelos complejos (polinomios de alto grado), y esta asociada al ‘sobre-ajuste’ (overfitting).

Los modelos con alto sesgo son aquellos que frente a distintos datos provenientes de la misma población no modifican mucho su predicción aunque en general ese valor de predicción no suele ser aceptable. El sesgo está asociado a ‘sub-ajuste’ (underfitting) y se suele observar en modelos muy sencillos (por ejemplo modelos lineales).

Variance vs Bias

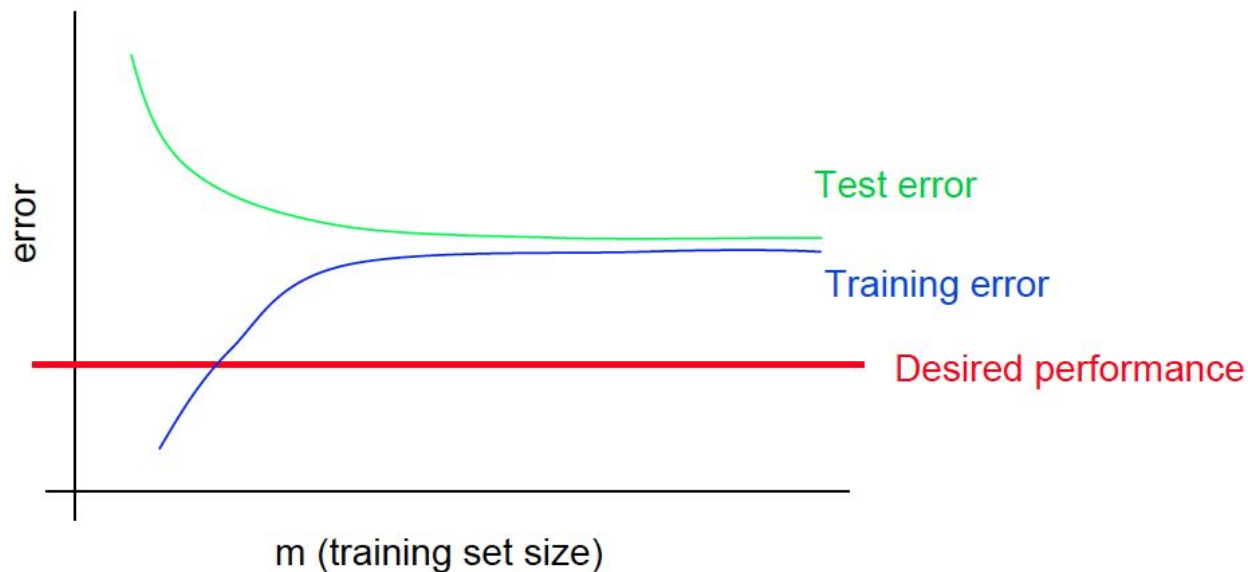


Learning curve: high variance



High variance means lower model complexity (like, less features)

Learning curve: high bias



High bias means higher model complexity (aka, too many features)

*Andrew Ng.

Momento del meme

Odio la Estadística 



@Odioestadistica



@Odiolaestadistica

Ejercicio

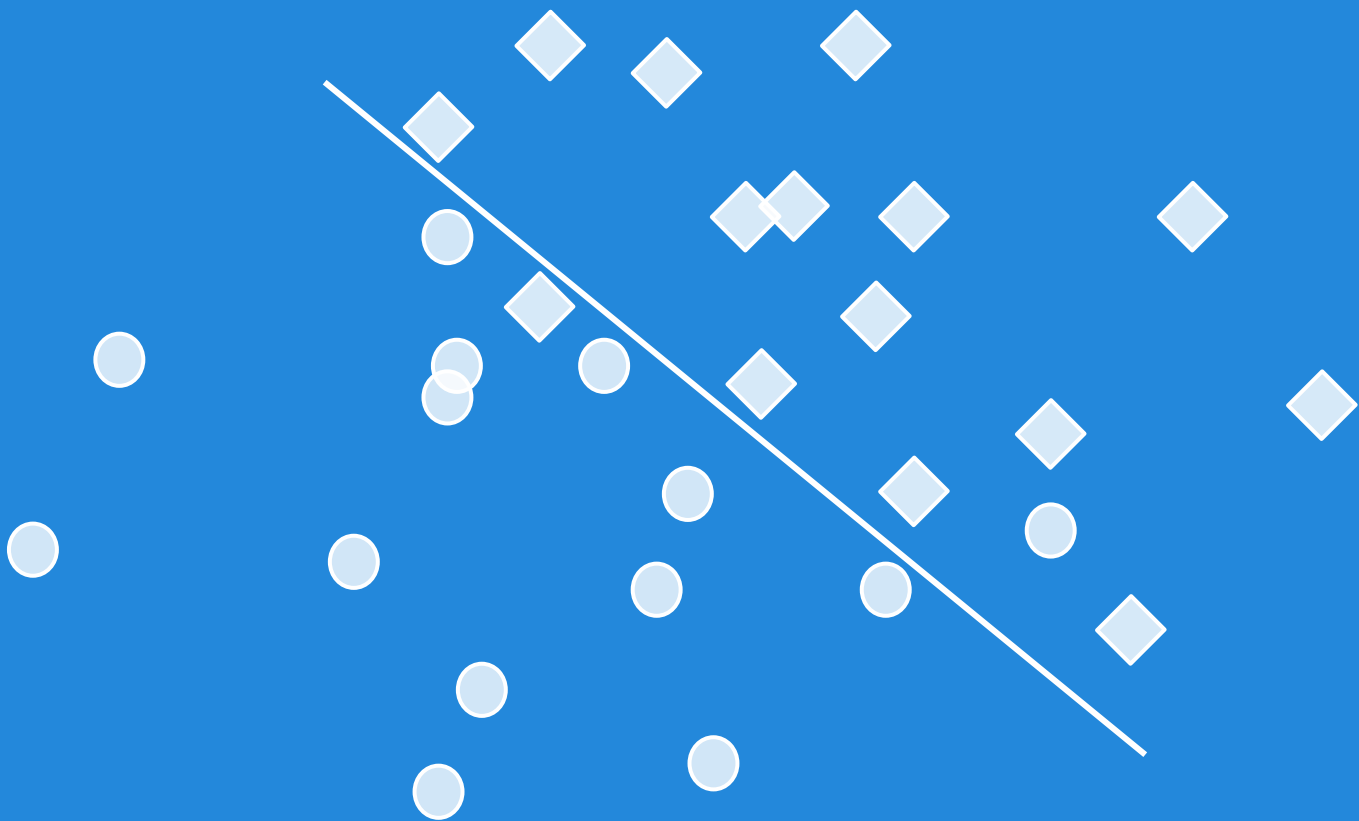
$$\mathbf{x}_{train} = \begin{matrix} & x_1 & x_2 \\ \begin{bmatrix} 1 & 1 \\ 3 & 0.5 \\ 2 & 3 \\ 2.5 & 1.5 \\ 1.5 & 2 \\ 2.5 & 4 \\ 3.5 & 1.5 \\ 4 & 3 \end{bmatrix} & \mathbf{y}_{train} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \end{matrix}$$

$$f(x) = w^T x + b$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$n = 8 \quad d = 2 \quad \mathcal{X} \in \mathbb{R}^2$$

Dado el dataset (x,y) de entrenamiento, calcular cuáles deberían ser los valores de los parámetros \mathbf{w} y \mathbf{b} para obtener una función lineal de decisión que clasifique los datos con el menor error. Graficar la solución.



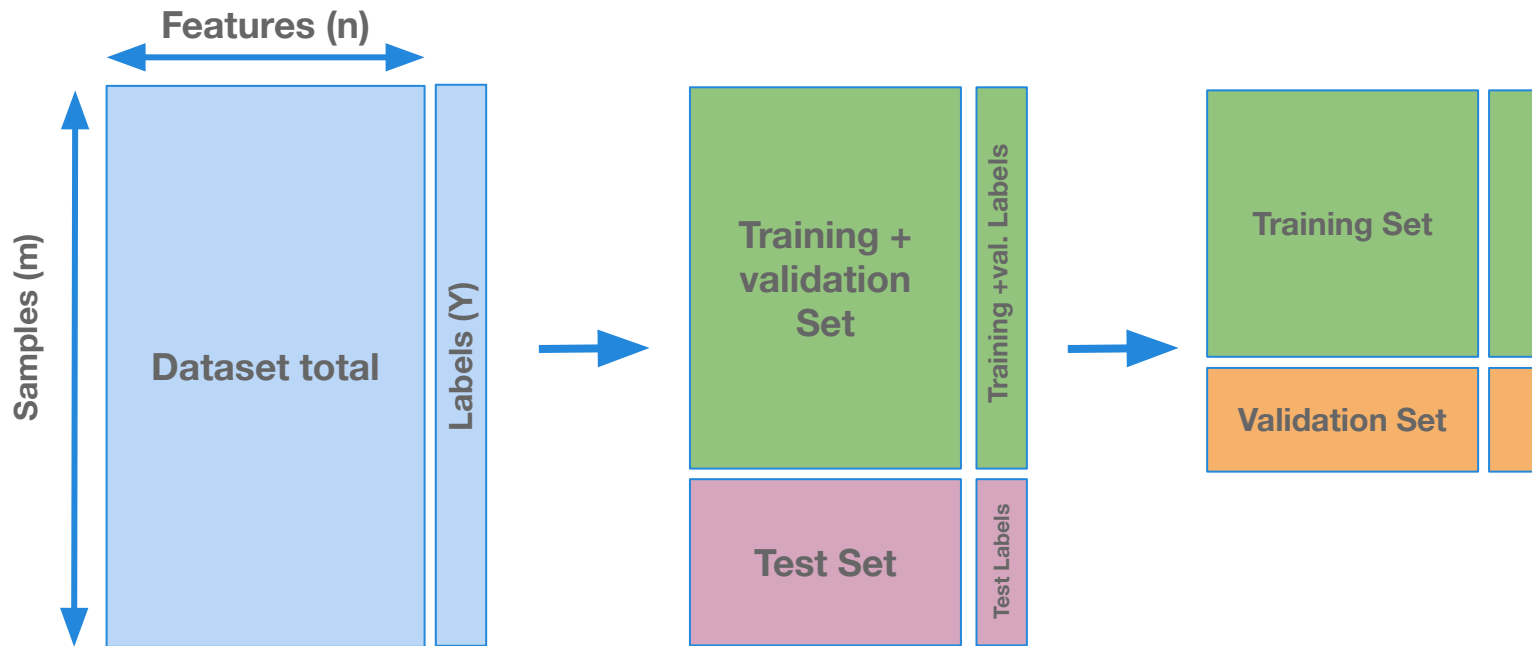
Aprendizaje supervisado: cross validation & hyperparameter tuning

Hiper-parametros

$f_{w,\lambda}(x)$ funcion de decision
 w parametros
 λ 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 utiles 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.



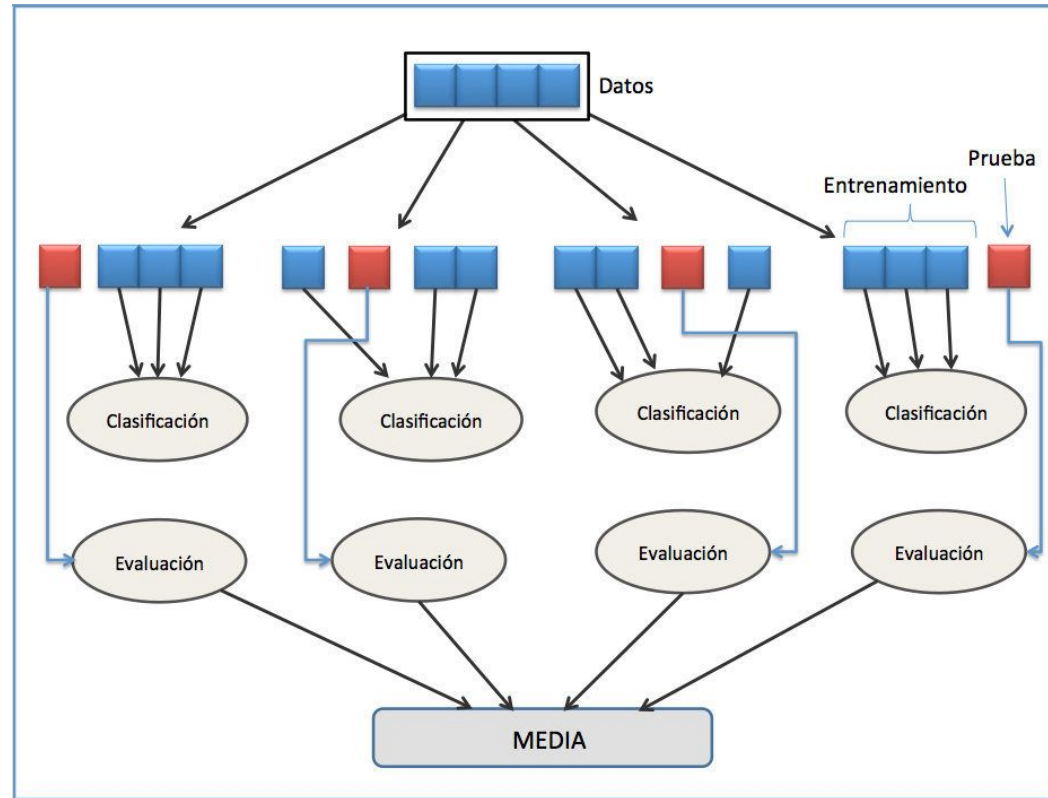
El clasificador aprenderá la regla de decisión utilizando el train set (samples + labels). Luego clasificará las muestras de test (sin mirar las labels de test) y se medirá la exactitud de clasificación en testeo.

Cross - Validation en training set

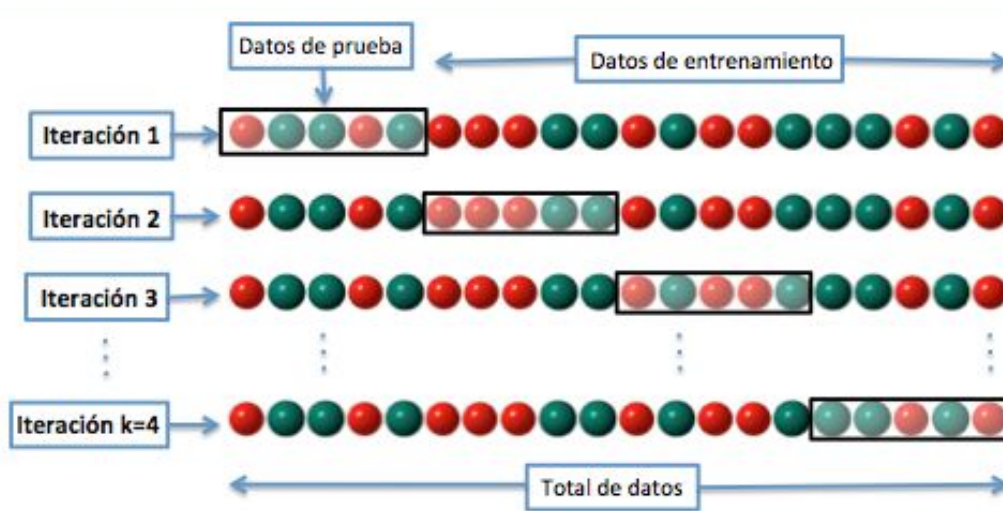
“**Cross validation**” (CV) se realiza con las muestras de entrenamiento. Consiste en dividir nuestro training set en **K folds** (K porciones) e iterar K veces.

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 clasificación con validación. Luego se realizará un promedio de la exactitud de clasificación 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.



Cross Validation



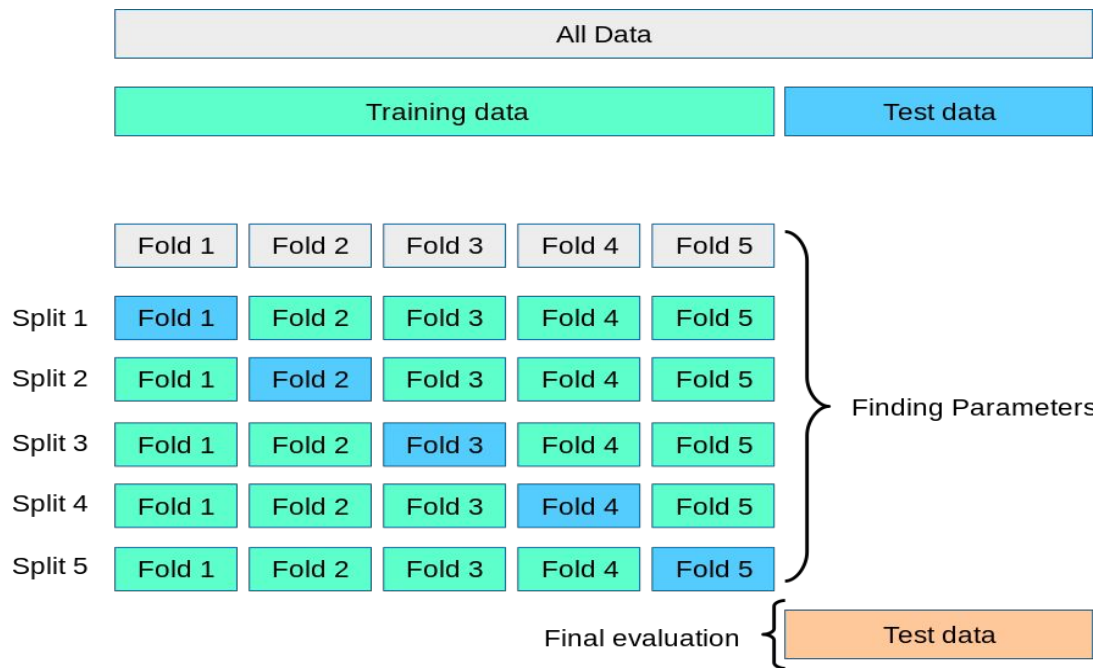
Veremos que nuestros modelos clasificadores a entrenar tendrán distintos “hiper-parámetros a elegir”, por ejemplo si deseamos que nuestro modelo clasificador sea complejo o no, o si queremos que sea “exigente” con cada error de clasificación o permisivo en mayor o menor grado.

Estos hiper-parámetros son elegidos inicialmente por el usuario. Luego por validación cruzada determinaremos el mejor hiper-parametro para clasificar.

Cross Validation

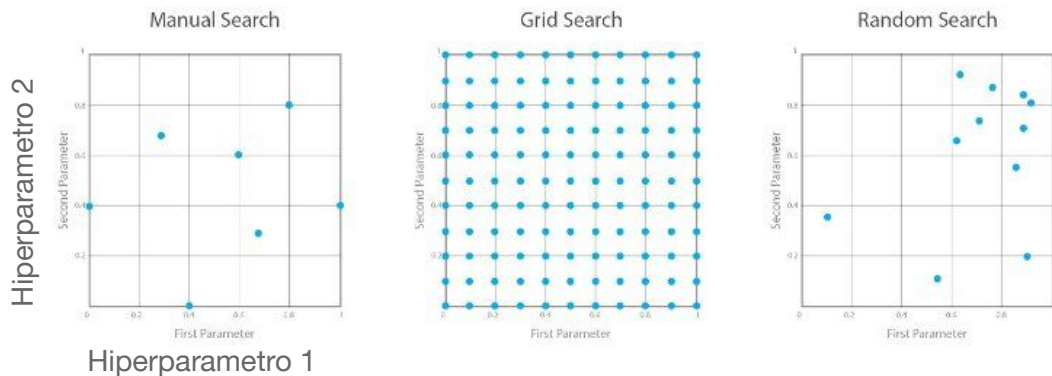
El cross validation también nos da una idea de como funciona el modelo propuesto frente a distintas particiones train-valid de mis datos. En general un buen modelo debería “clasificar aceptablemente bien” en todas las particiones.

Es una manera de “sincerar” si el modelo funciona bien en distintos escenarios y no solo depender de la suerte de nuestra partición.



Grid Search

- Los modelos de clasificación que utilizaremos consistirán de **hiper-parámetros** que el usuario debe seleccionar. Estos determinarán la regla de decisión y por ende la performance del modelo.
- Para saber qué hiper-parámetros seleccionar, lo que haremos es generar una lista de los mismos y probaremos todas las combinaciones posibles de ellos (**grid-search**)
- La combinación de hiper-parámetros que mayor **Train Accuracy** promedio genere durante el cross-validation será la que usaremos para testear el modelo.



Pipeline: Train, Validate, Test Model

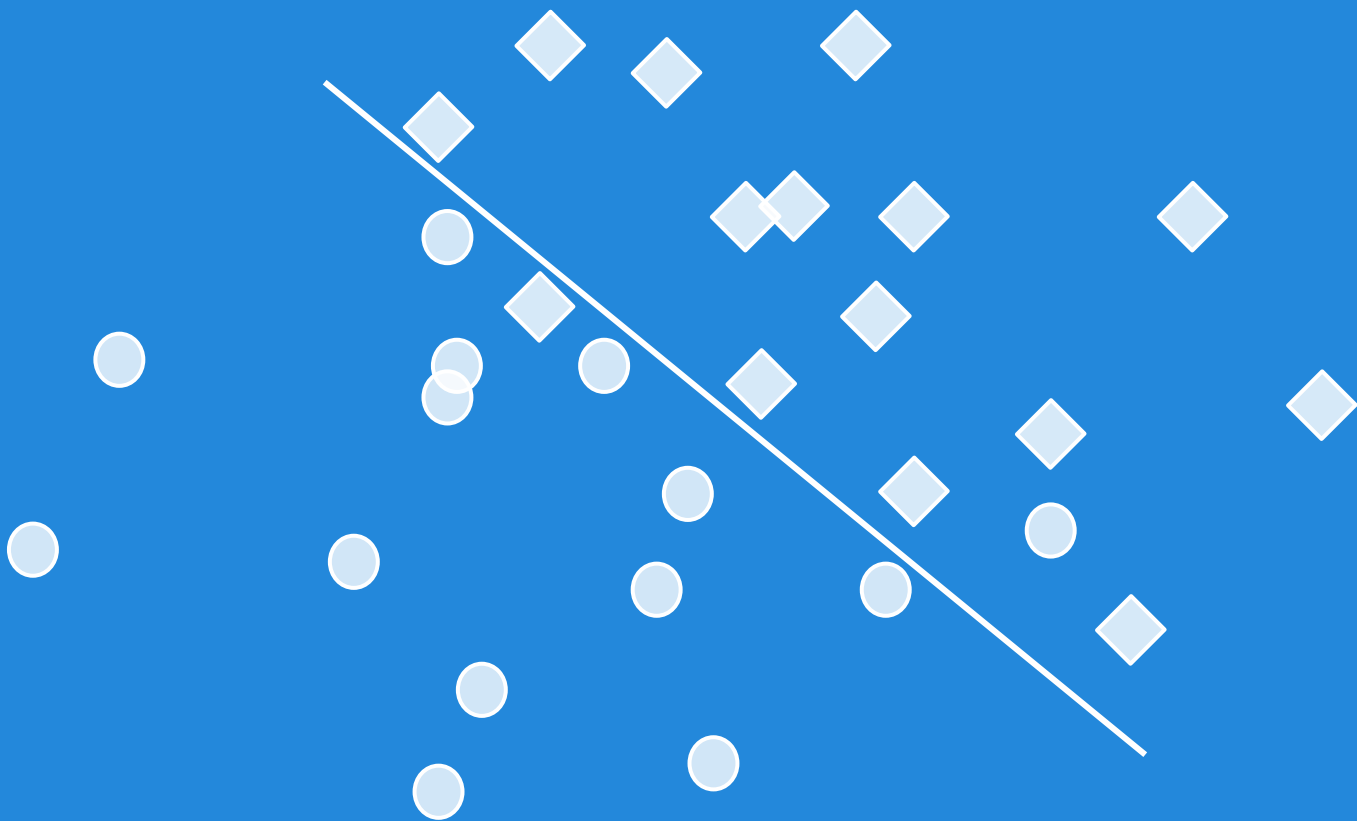
**Dividir
Train y
Test**

**Cross Validation &
Grid Search con
Train Set (utilizando
Xtrain e Ytrain)**

**Selección del
mejor modelo e
hiperparametro**

**Clasificar muestras
de Test (xtest) sin
mostrarle al modelo
las Ytest.**

**Evaluar
resultados de
clasificación en
test (comparar
Ypred vs Ytest)**



Aprendizaje supervisado: mediciones de desempeño/performance en clasificación

classification results: confusion matrix

		Predicted Label	
		Class1 (-)	Class2 (+)
True Label	Class1 (-)	True Negative	False Positive
	Class2 (+)	False Negative	True Positive

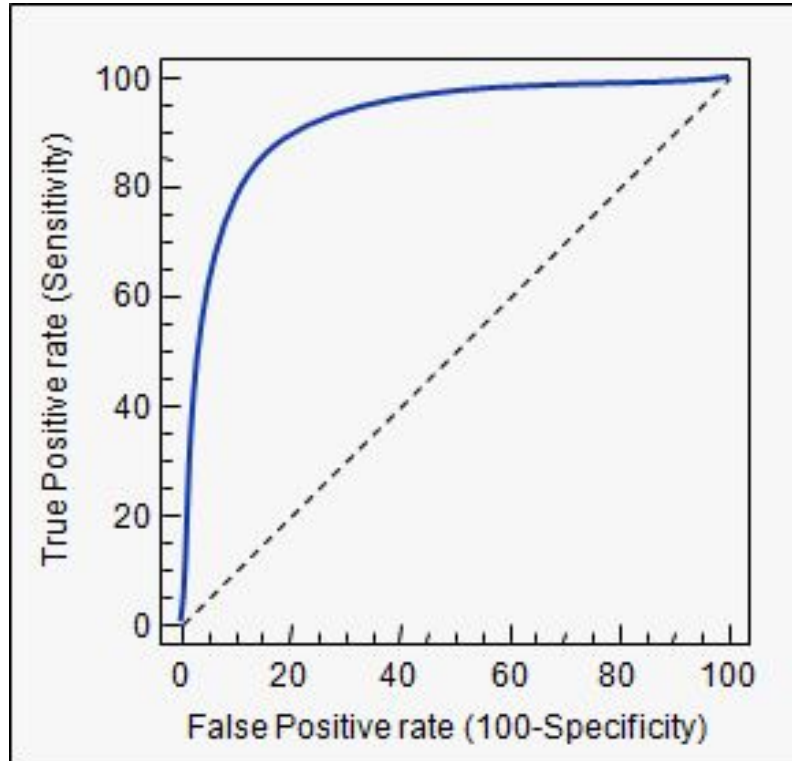
Accuracy = $(TN+TP) / \text{Total}$

Sensitivity (recall) = $TP/(TP+FN)$

Specificity = $TN/(TN+FP)$

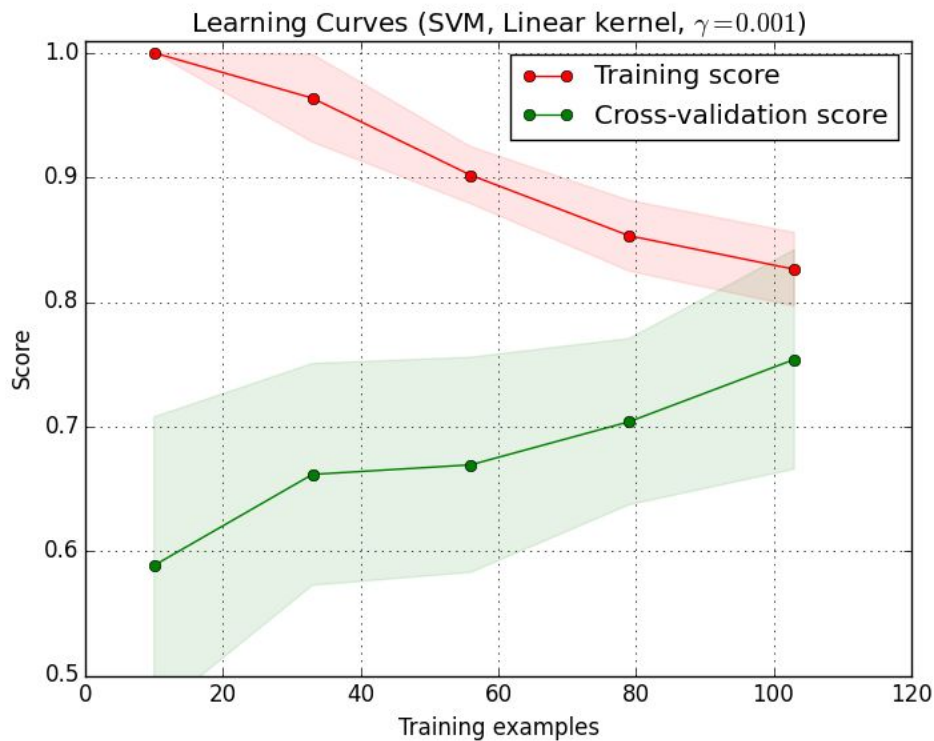
La matriz de confusión es un elemento para evaluar los resultados de la clasificación. En cada posición se cuentan los TP, TN, FP, FN. Luego se obtienen los coeficientes de accuracy, Sensitivity, specificity.

classification results: AUC ROC



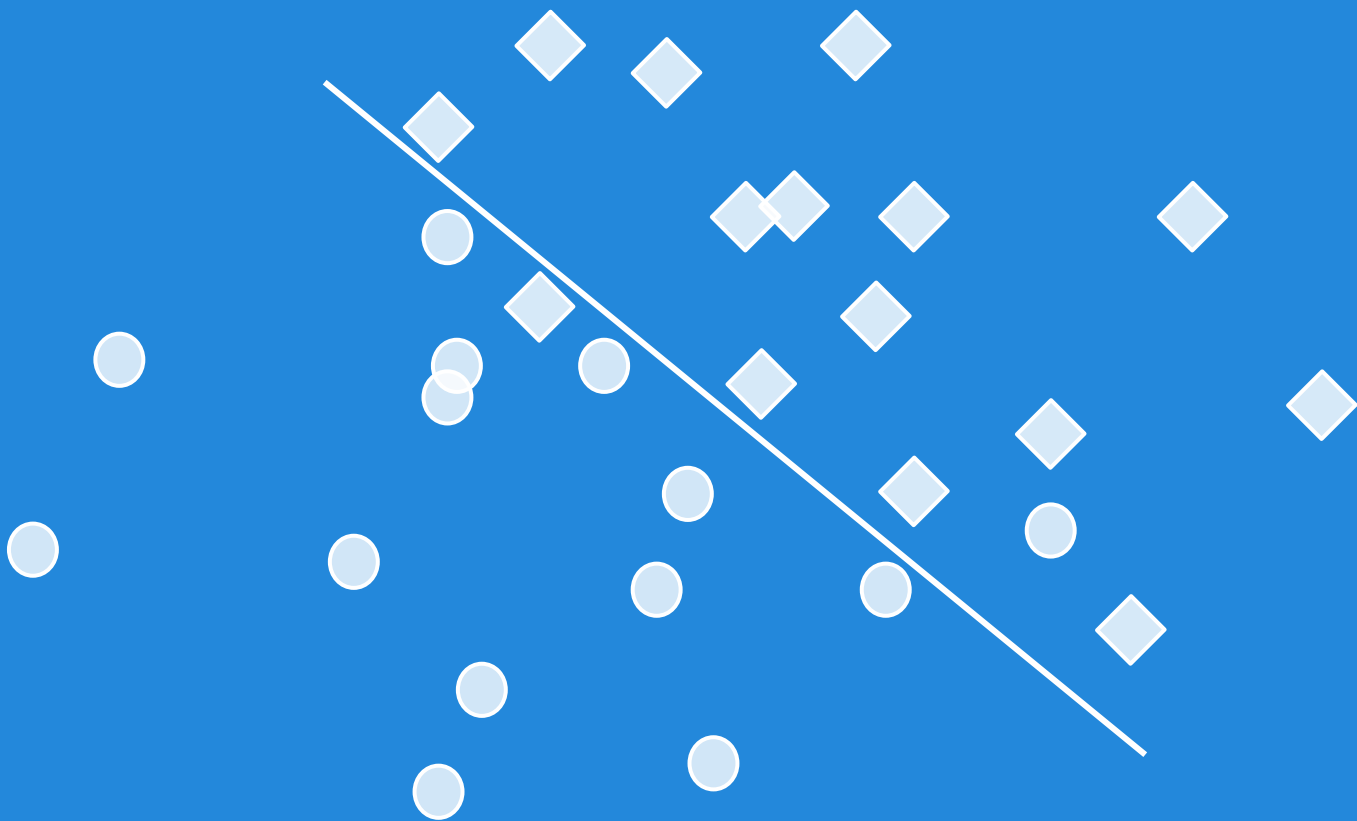
- El área bajo la curva ROC (AUC) da una idea de cuan bueno es mi clasificador independientemente del accuracy.
- Tomando una sola clase, se analizan distintos umbrales de clasificación y se contempla la relación entre TP y FP.
- Cuanto más cerca de 0.5 sea el AUC, más similar a “arrojar una moneda” será mi clasificador. Cuanto más cerca de 1 sea el área bajo la curva, mejor mi clasificador.

Learning Curve



Podemos evaluar el funcionamiento del clasificador con distintas cantidades de muestras de entrenamiento, de menos a más.

La curva de aprendizaje es aquella que se forma al computar el accuracy en train y test en función de la cantidad de muestras de train. Idealmente debería converger a un valor determinado.



Aprendizaje supervisado: modelos de clasificación

Clasificadores

Vamos a estudiar tres modelos de clasificación

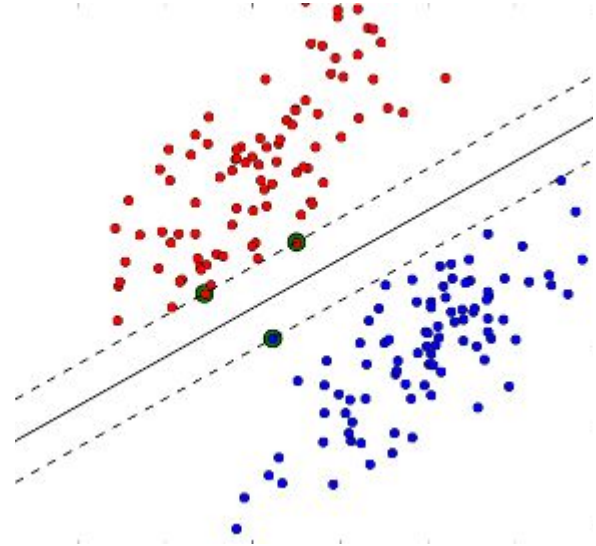
- Support Vector Machines
- K-Nearest Neighbors
- Logistic Regression

Cada uno tendrá ventajas y debilidades respecto a los otros. Hay decenas de clasificadores que por razones de tiempo no incluimos en este curso.

Support Vector Machines

Support Vector Machines

- Clasificador Lineal.
- Busca el hiperplano separador que **maximiza** el margen entre clases.
- Cuando las clases no son separables linealmente se acude al “soft-margin”, penalizador que permite muestras “del otro lado”.
- Cada muestra mal clasificada es penalizada por un **costo C** que el usuario selecciona (un ej. de hiper-parámetro).
- El margen separador queda definido por “s” muestras. **Estas muestras son llamadas support vectors.**

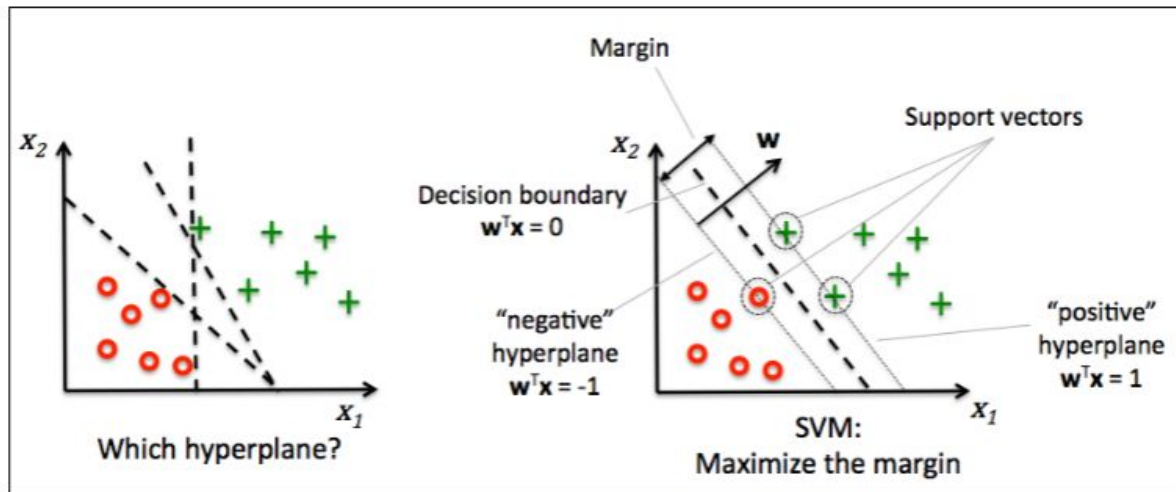


Classification Models: SVM “Kernel Trick”

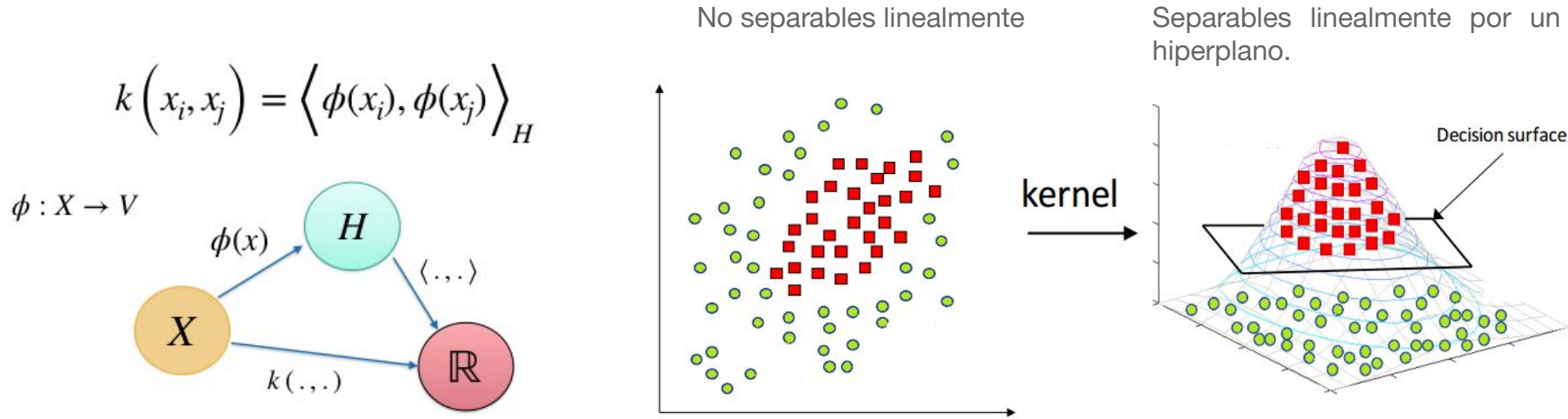
El hiperplano calculado por SVM será el mejor de todas las opciones posibles -> tiene **solución convexa** con un máximo global :)

La manera de lidiar con clases superpuestas es por medio de un “**soft-margin**” que permite clasificar mal con cierta penalización.

El margen obtenido siempre es el máximo. El hiper-plano estará determinado por los “support vectors”.



Classification Models: SVM “Kernel Trick”



Los kernels son funciones de similitud entre muestras. Mapean nuestros datos a un espacio de alta dimensión donde son linealmente separables. Allí en ese nuevo espacio donde son mapeadas las muestras se aplican los productos internos (o similitud). Cuando usamos SVM, podemos aplicar un kernel para facilitar la clasificación, es decir que el hiperplano estará afectado por el kernel.

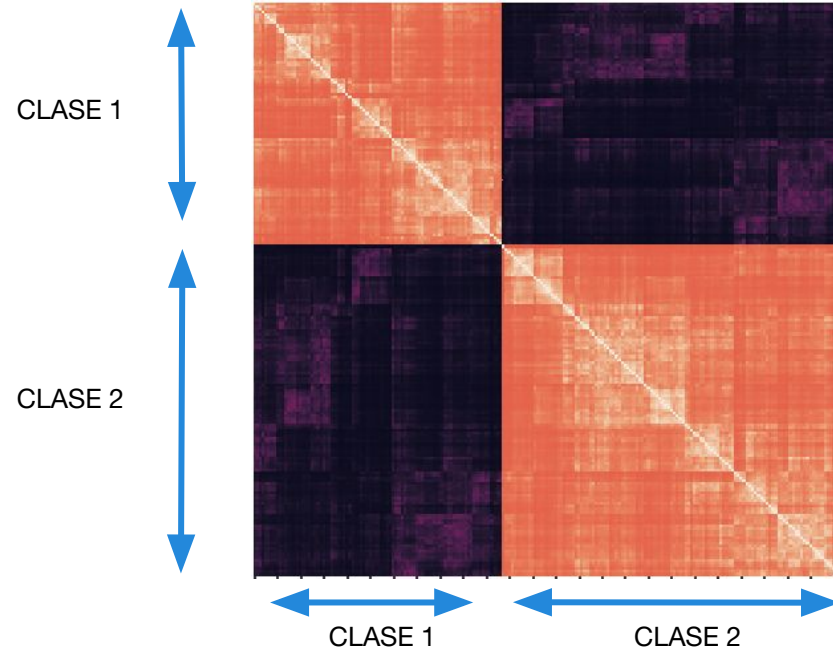
Classification Models: SVM “Kernel Trick”

¿qué es un buen kernel? En un contexto supervisado, sería aquella función de similitud que mapea mis muestras de la misma clase muy cerca entre si y al mismo tiempo mapea lejos las muestras de distintas clases.

El kernel tomará un valor de 0 si son de distintas clases (ortogonales) y tomará el valor de 1 si son muestras iguales o de la misma clase (todo esto idealmente).

Recordemos que el kernel es una función que calcula los productos internos entre muestras.

En la figura, las muestras de la misma clase tienen valores claros (cerca de 1) y las de distinta clase valores cercanos a 0 (oscuros).



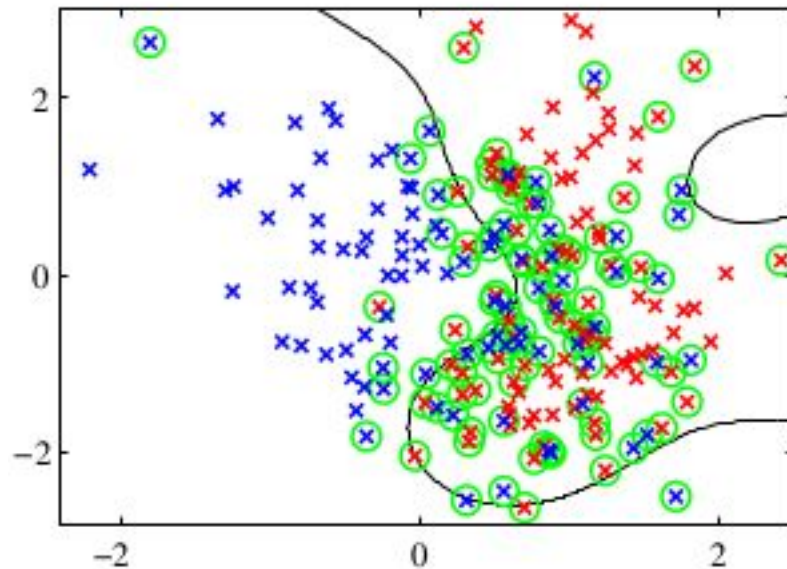
$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_H$$

SVM: clases superpuestas y frontera no-lineal

En la mayoría de los problemas reales las clases están superpuestas por lo que no es posible separarlas con un hiperplano.

Mas aún, a pesar de utilizar un kernel, la distribución de las clases puede tener cierta complejidad que no permita una clasificación perfecta (la elección del kernel no es trivial!)

En la figura los puntos de verde son los “support vectors”.



Non-linear boundary using gaussian kernel. *Bishop, Pattern Recognition



KERNELS

HHD
HISTORY.COM

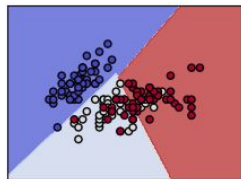
SVM: Hiper Parametros

Kernels más frecuentes: Gaussian, Linar, Polynomial

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

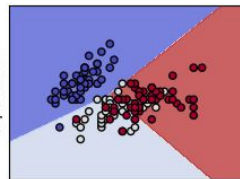
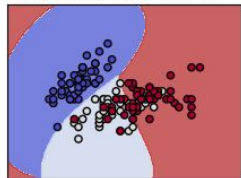
$$K_{lin}(x_i, x_j) = \langle x_i, x_j \rangle$$

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



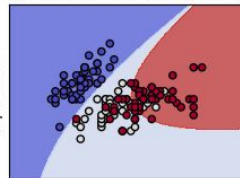
Sepal length

SVC with RBF kernel



Sepal length

SVC with polynomial (degree 3) kernel



Cada kernel hará que el SVM genere distintos tipos de frontera de clasificación. Los kernels gaussianos y polinomiales generarán fronteras no lineales mas complejas y el lineal o el polinomial de bajo grado mas sencillas. ¿cuando usamos un kernel complejo y cuando uno sencillo?

SVM: Hiper Parametros

El SVM puede generar una familia de funciones discriminantes. La función final quedará determinada por los hiper-parámetros que seleccionemos (via cross validation)

- Tipo de Kernel = Lineal, Polinomial, Gaussiano
- C = “Costo” por mal clasificación (todos los kernels)
- γ = Kernel Gaussiano (RBF)
- Degree = Kernel Polynomial

Support Vector Machines



[Home](#) [Installation](#) [Documentation](#) [Examples](#)

Google Custom Search



Previous Next Up
1. Kernel c... 1.5. Stochastic... 1. Supervised...

scikit-learn v0.19.2

[Other versions](#)

Please [cite us](#) if you
use the software.

4. Support Vector Machines

4.1. Classification

1.4.1.1. Multi-class classification

1.4.1.2. Scores and probabilities

1.4.1.3. Unbalanced problems

4.2. Regression

4.3. Density estimation, novelty detection

4.4. Complexity

4.5. Tips on Practical Use

4.6. Kernel functions

1.4.6.1. Custom Kernels

- 1.4.6.1.1. Using Python functions as
kernels

- 1.4.6.1.2. Using the Gram matrix

- 1.4.6.1.3. Parameters of the RBF
Kernel

4.7. Mathematical formulation

1.4.7.1. SVC

1.4.7.2. NuSVC

1.4.7.3. SVR

4.8. Implementation details

1.4. Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for [classification](#), [regression](#) and [outliers detection](#).

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different [Kernel functions](#) can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing [Kernel functions](#) and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see [Scores and probabilities](#), below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

1.4.1. Classification

SVM en Python con Scikit-Learn



```
# Samples and features
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

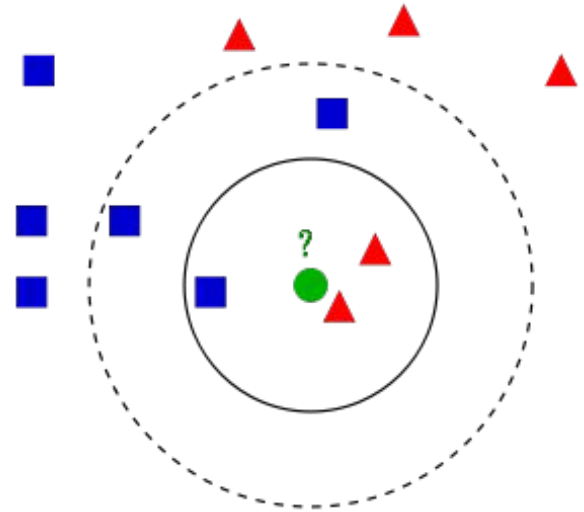
# Labels (two classes)
y = np.array([1, 1, 2, 2])

# support vector classification
from sklearn.svm import SVC
# define SVM model
svm = SVC(gamma=1, C = 1, kernel='rbf')
# fit SVM on training data
svm.fit(X, y)
# predict a new independent sample
y_prediction = clf.predict([[-0.8, -1]])

print(y_prediction)
[1]
```

Classification Models: KNN

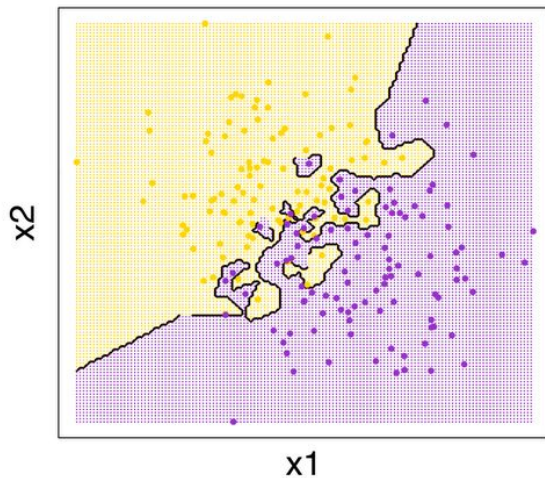
- Clasifica cada nuevo dato en el grupo que corresponda, según tenga **K** vecinos más cerca de un grupo o del otro.
- Calcula la **distancia** del elemento nuevo a cada uno de los existentes y ordena esas *distancias* para seleccionar a qué grupo al que pertenece.
- Selecciona la etiqueta (Y) que más frecuente aparece entre las K clases para “etiquetar” a la muestra en cuestión.
- Hiper-parámetro: “K” vecinos



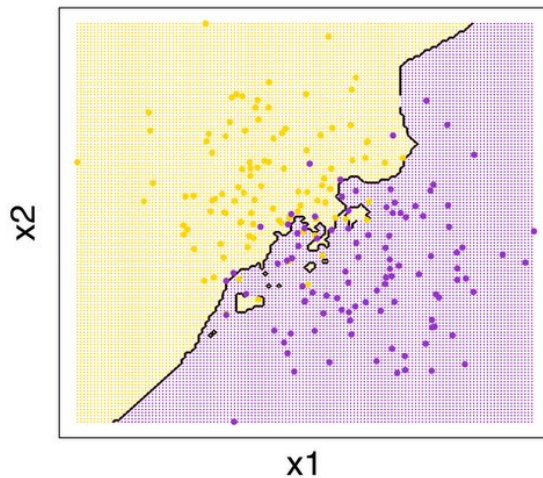
$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

KNN: variar el parametro K

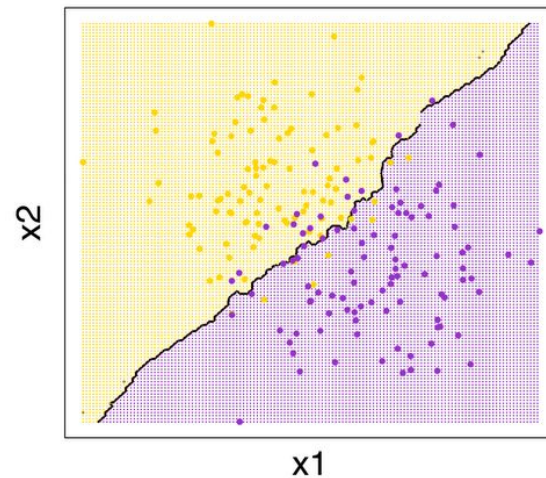
Binary kNN Classification (k=1)



Binary kNN Classification (k=5)



Binary kNN Classification (k=25)



Classification Models: KNN



[Home](#) [Installation](#) [Documentation](#) [Examples](#)

Google Custom Search



[Previous](#)
sklearn.neigh

[Next](#)
sklearn.neigh

[Up](#)
API
Reference

scikit-learn v0.21.3

[Other versions](#)

Please **cite us** if you
use the software.

`sklearn.neighbors.KNeighborsClassifier`

Examples using

`sklearn.neighbors.KNeighbors`

`sklearn.neighbors.KNeighborsClassifier`

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

[\[source\]](#)

Classifier implementing the k-nearest neighbors vote.

Read more in the [User Guide](#).

Parameters: `n_neighbors` : *int, optional (default = 5)*

Number of neighbors to use by default for `kneighbors` queries.

weights : *str or callable, optional (default = 'uniform')*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, optional (default = 30)*

Leaf size passed to `BallTree` or `KDTree`. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *integer, optional (default = 2)*

Power parameter for the Minkowski metric. When $p = 1$, this is equivalent to using `manhattan_distance` (L1), and `euclidean_distance` (L2) for $p = 2$. For arbitrary p , `minkowski_distance (l_p)` is used.

KNN en Python con Scikit-Learn



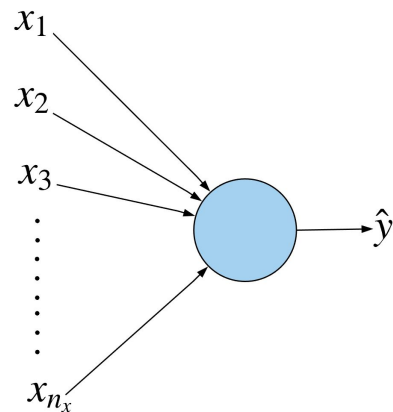
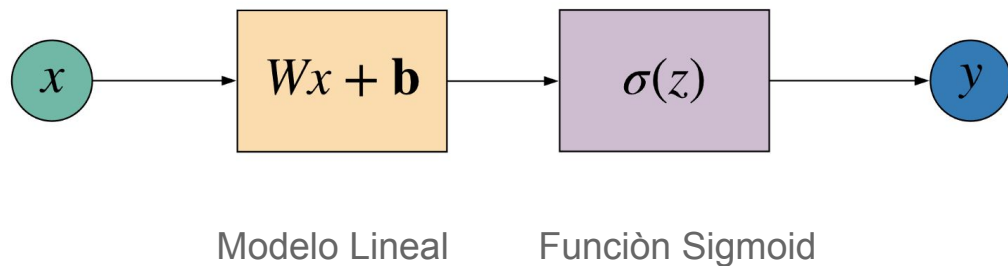
```
# Samples and features
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

# Labels (two classes)
y = np.array([1, 1, 2, 2])

# import KNN model
from sklearn.neighbors import NearestNeighbors
# define KNN model
knn = NearestNeighbors(n_neighbors=1)
# fit SVM on training data
knn.fit(X, y)
# predict a new independent sample
y_prediction = knn.predict([[-0.8, -1]])
# print prediction
print(y_prediction)
[1]
```

Logistic Regression

- Logistic regression es un clasificador lineal.
- Es una regresión lineal precedida de una función de activación “sigmoid”, lo que genera que el output sea binario y no continuo como una regresión normal.
- Puede entenderse como una red neuronal de una sola capa y una sola neurona.
- A cada muestra clasificada, le asigna una probabilidad de pertenecer a cada clase existente en el problema. Si la probabilidad es mayor a cierto threshold (0.5) entonces pertenece a una clase y viceversa.



Logistic Regression

El regresor logístico debe aprender un parámetro interno (no es hiper-parámetro) por cada dimensión del vector de entrada (vector W). Para eso calculará el gradiente del error de clasificación y tratará de minimizarlo.

Probabilidad de la clase Y_i dado un vector de entrada x .

$$p(y_i|x) = \sigma(w^T x)$$

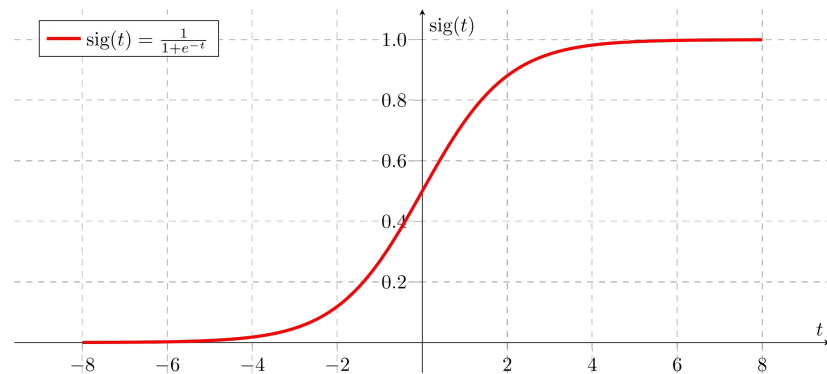
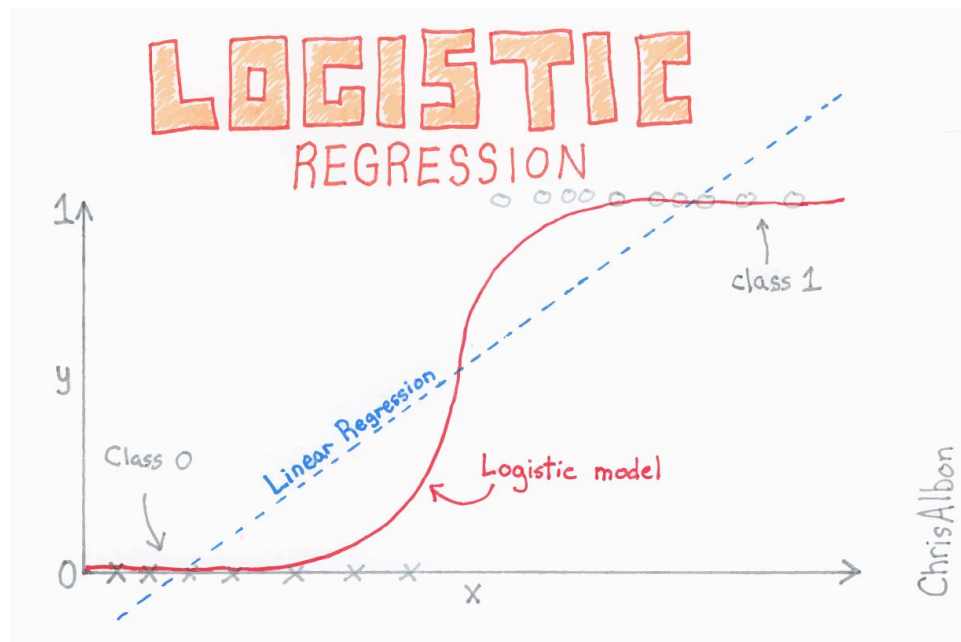
Función Sigmoid

La suma de las probabilidades de pertenecer a cada clase debe sumar 1.

$$p(y_1|x) = 1 - p(y_2|x)$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Logistic Regression



Función de activación “sigmoid”: mapea cualquier valor de X a un valor entre 0 y 1 pero nunca llega a estos extremos.

$$f(x) = \frac{1}{1 - e^{-x}}$$

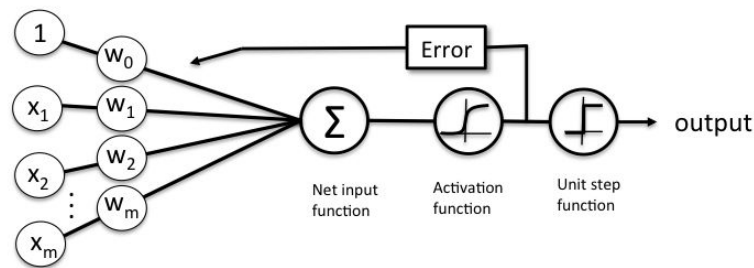
Logistic Regression

LOGISTIC VS. **LINEAR**
REGRESSION REGRESSION

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + \dots + b_k x_k)}}$$

Logistic model Linear model

Chris Albon



Schematic of a logistic regression classifier.

Lo que “aprende” logistic regression es el vector de “pesos” \mathbf{W} de cada variable. Queremos que ese vector tenga “poder de generalización” para que clasifique bien nuevas muestras independientes una vez entrenado.

LR: Hiper Parametros

Los hiper-parámetros de Logistic Regression son

- C = “Costo” , al igual que SVM, es un penalizador que regulariza la solución computando clasificaciones erróneas.
- Penalizador L1 o L2: aplica una penalización bajo la norma L1 o L2 al vector \mathbf{W} . De esta manera evita que el modelo quiera “sobre-ajustarse” a los datos de entrenamiento. En otras palabras, evita que existan posiciones de W muy muy altas y otras casi nulas, o viceversa, que solo algunas posiciones de W se activen y el resto no.

LR en Python con Scikit-Learn



```
# Samples and features
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

# Labels (two classes)
y = np.array([1, 1, 2, 2])

# import LR model
from sklearn.linear_model import LogisticRegression
# define KNN model
lr = LogisticRegression(random_state=0, solver='lbfgs')
# fit SVM on training data
lr.fit(X, y)
# predict a new independent sample
y_prediction = lr.predict([[-0.8, -1]])
# print prediction
print(y_prediction)
[0]
```

Manos a la obra



JAKE-CLARK.TUMBLR