

DATA SCIENCE

Modelos de Aprendizaje Supervisado



UCEMA

Tipos de Aprendizaje

Aprendizaje Supervisado

El aprendizaje supervisado es aquel que para un conjunto de datos de entrada conocemos de antemano el Target Objetivo. Consta de 2 fases, una de entrenamiento y otra de pruebas

Aprendizaje No Supervisado

En el aprendizaje no supervisado para un conjunto de datos de entrada, NO conocemos de antemano los datos de salida. Se buscan similitudes y correlaciones entre los datos.

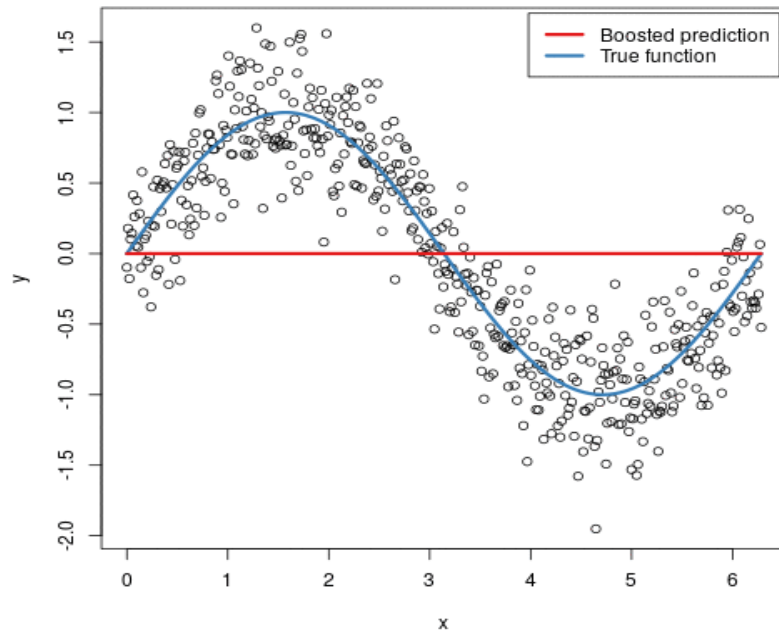
Reinforce Learning

Tipo de aprendizaje donde se repite la realización de una tarea para determinar qué acciones debe escoger en un entorno dado con el fin de maximizar "recompensa" o premio acumulado.

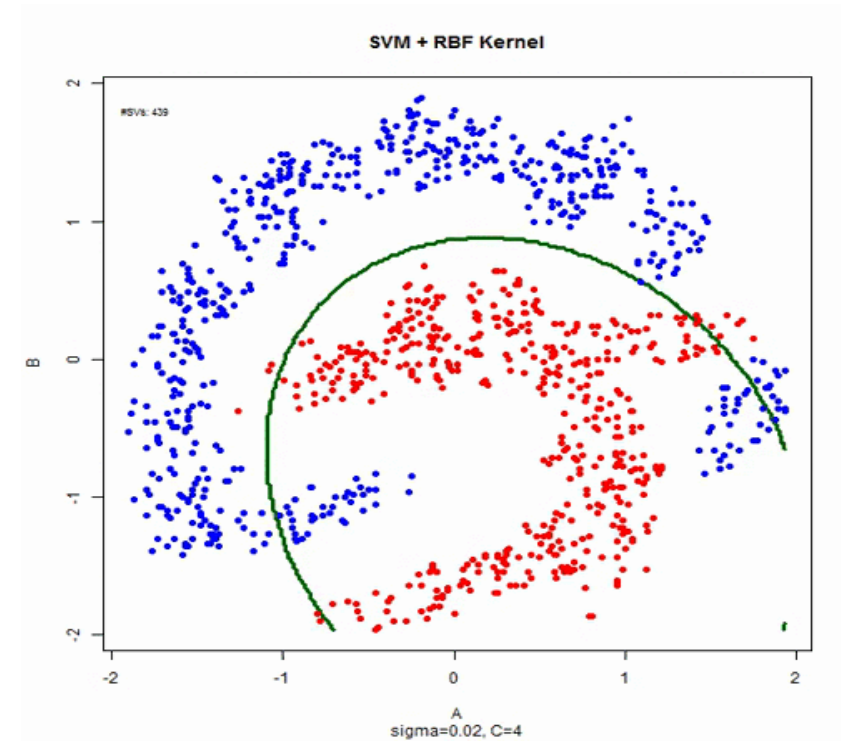
Metodología del Aprendizaje Supervisado



Tipos de Modelos



Regresión



Clasificación

Construcción de la Base

Tipos de Datos

Datos Estructurados

m2	baños	...	precio
140	4		350k
40	1		80k
⋮	⋮		⋮
80	2		180k

edad	deuda	...	default
41	0		0
18	8000		1
⋮	⋮		⋮
27	35000		0

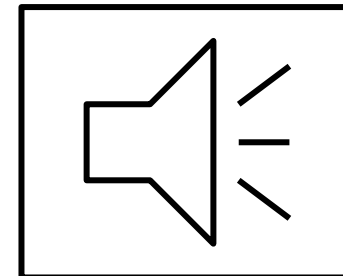
Datos No Estructurados



Imágenes

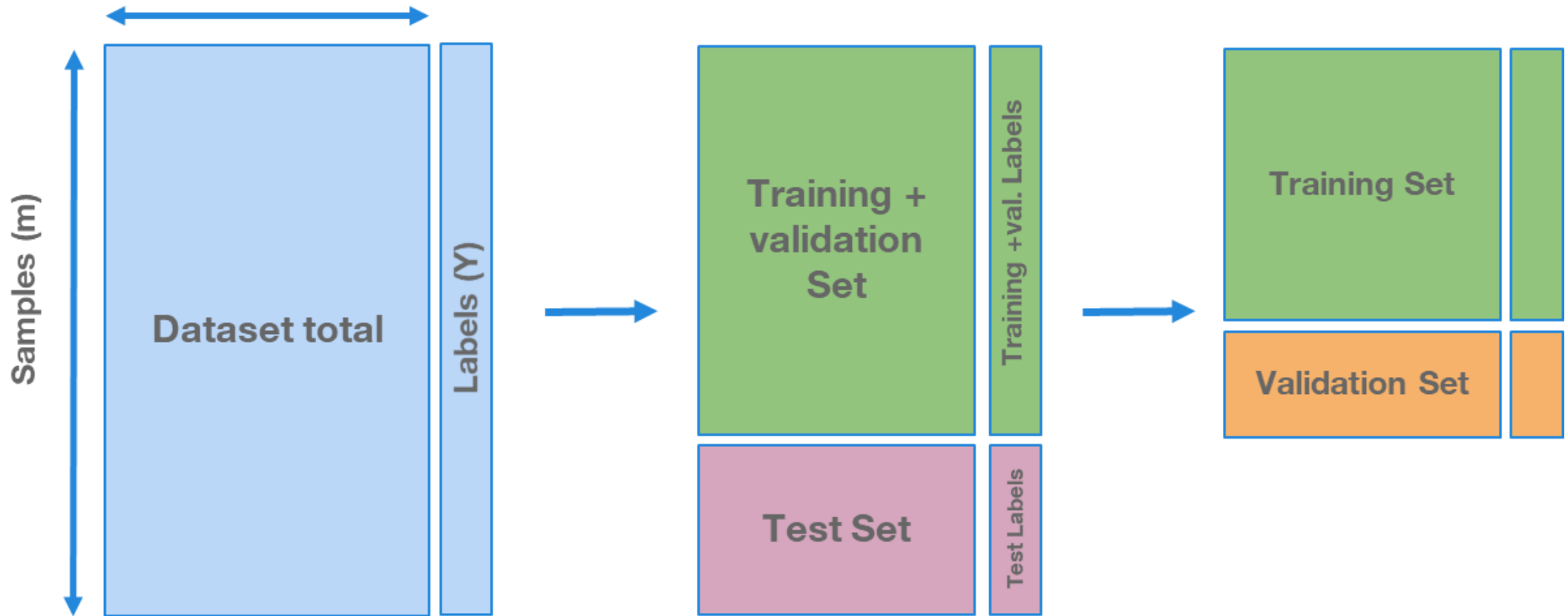
Hace mucho
tiempo en una
galaxia muy, muy
lejana...

Texto



Audio

Construcción de la Base



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.

Medición de Resultados

Para Clasificación: Matriz de Confusión

		Predicción	
		Clase 1	Clase 2
Verdadera	Clase 1	True Negative	False Positive
	Clase 2	False Negative	True Positive

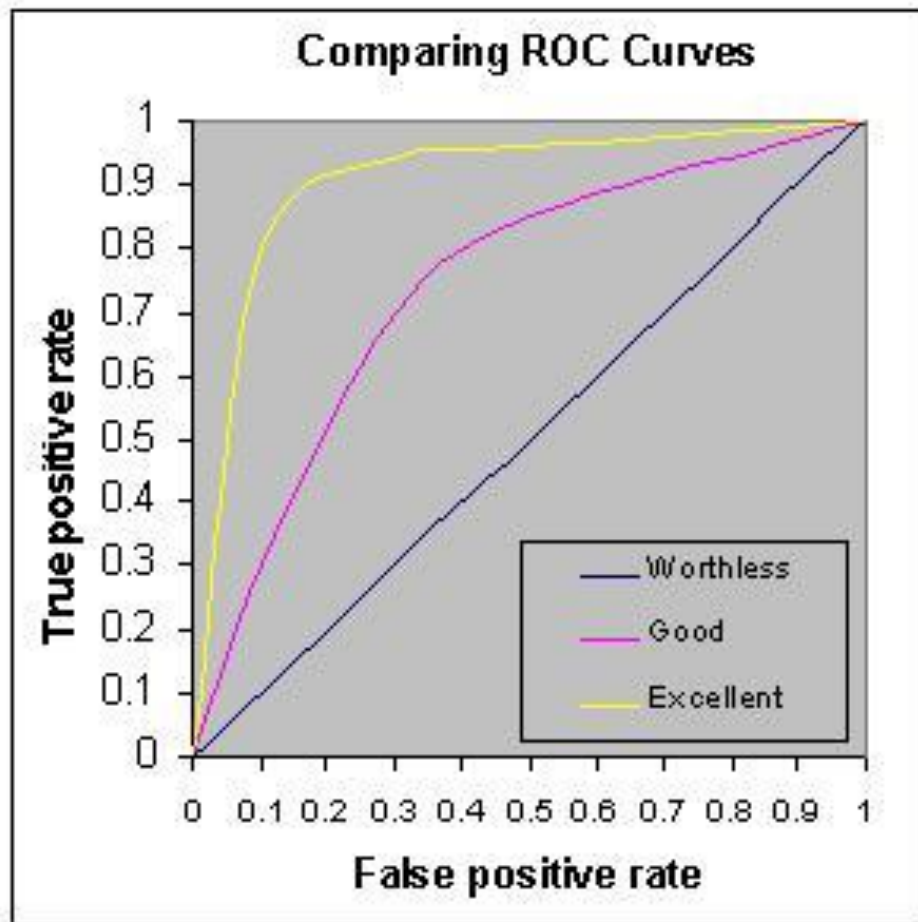
$$\text{Accuracy} = (\text{TN} + \text{TP}) / \text{Total}$$

$$\text{Error} = (\text{FP} + \text{FN}) / \text{Total}$$

$$\text{Recall} = \text{TP} / (\text{FP} + \text{TP})$$

$$\text{Precision} = \text{TP} / (\text{FN} + \text{TP})$$

Área bajo la curva ROC



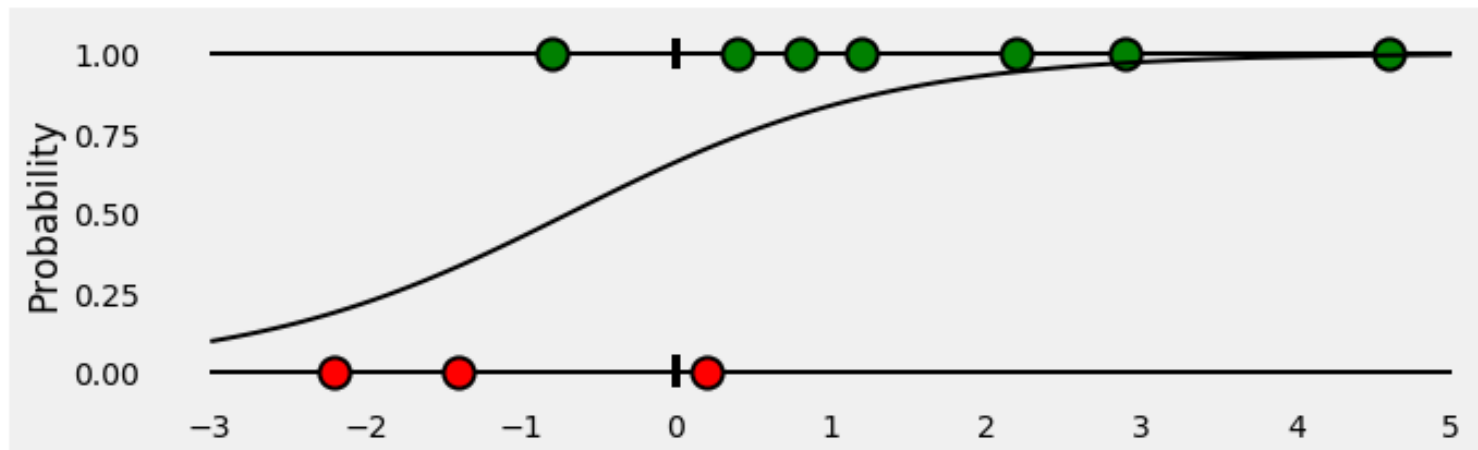
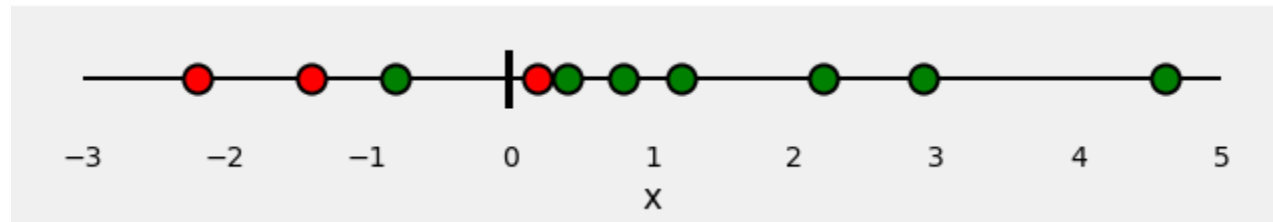
Y	Prob
1	0,9
1	0,88
1	0,84
0	0,83
1	0,8
1	0,77
1	0,75
0	0,75

Ejemplo Visual: <http://www.navan.name/roc/>

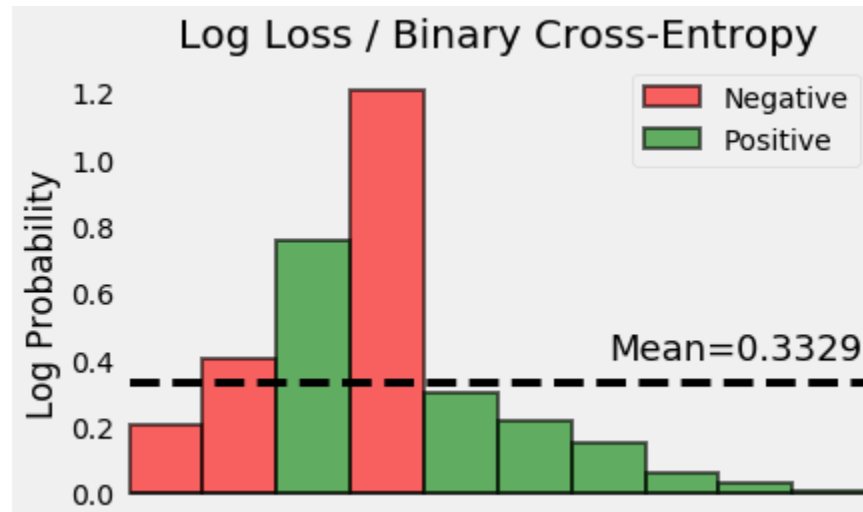
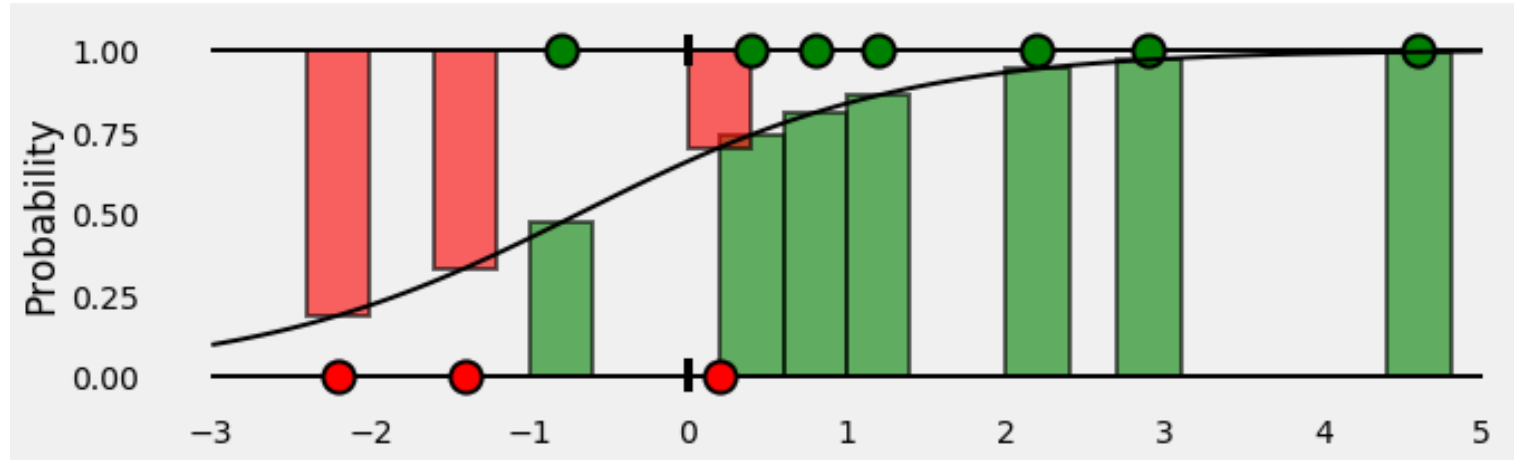
Log Loss - Entropía Cruzada Binaria

Binary Cross-Entropy

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N \underbrace{y_i \cdot \log(p(y_i))}_{\text{negative}} + \underbrace{(1 - y_i) \cdot \log(1 - p(y_i))}_{\text{positive}}$$



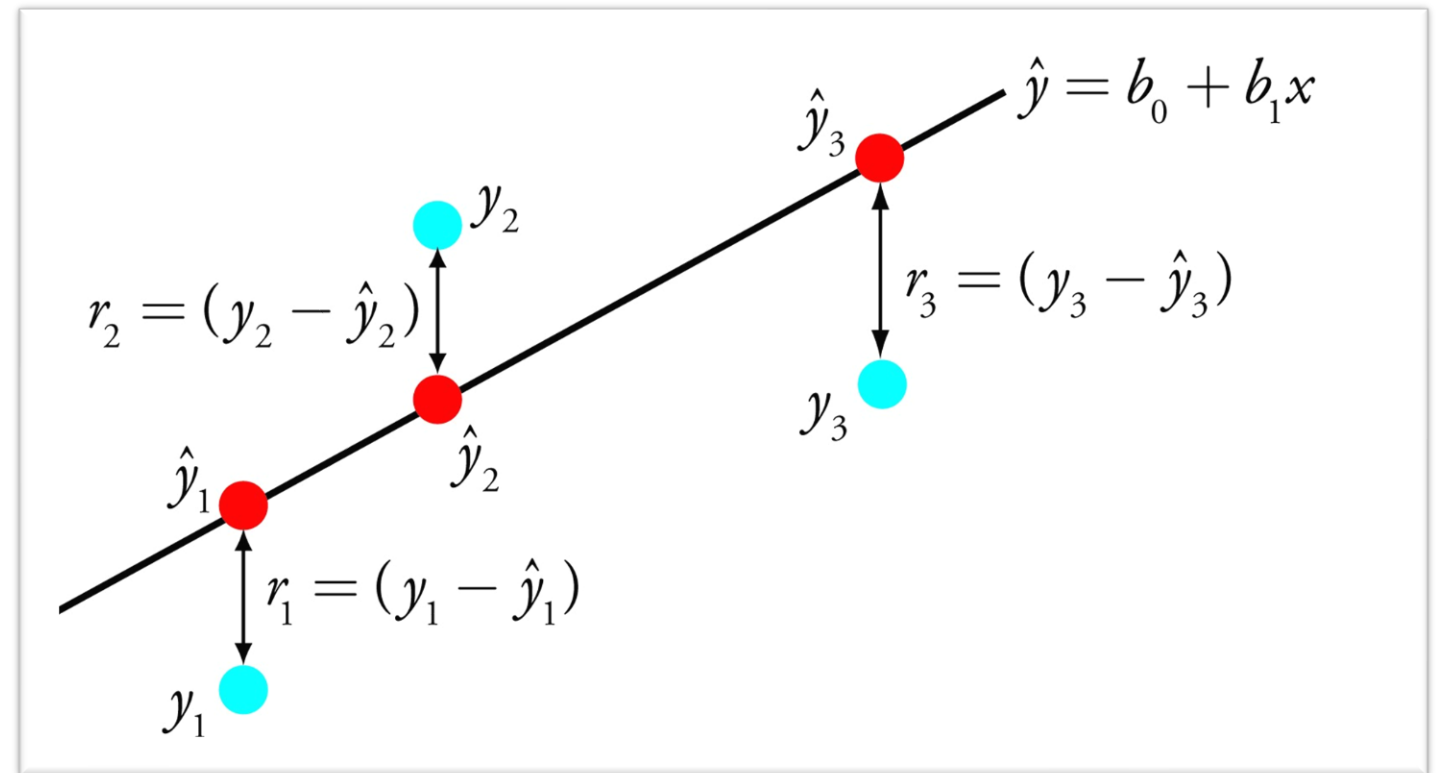
Log Loss - Entropía Cruzada Binaria



Para Regresión: Error Medio Cuadrático o Absoluto

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$



Regularización

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda f(\beta)$$

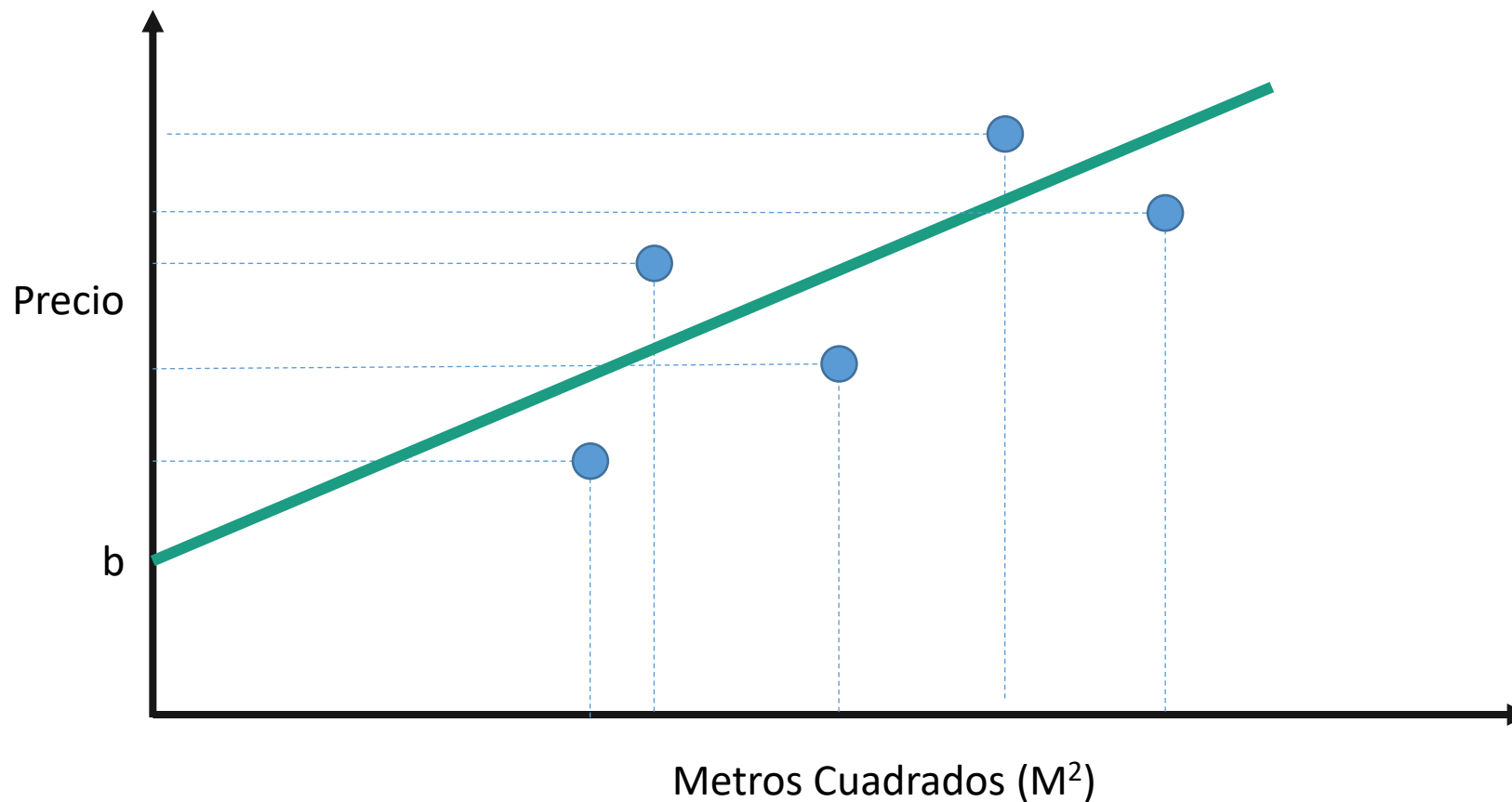
L1 (Lasso) $+ \lambda \sum_{i=1}^N |w_i|$

L2 (Ridge) $+ \lambda \sum_{i=1}^N w_i^2$

Modelos Básicos

Regresión lineal

Predecir el precio de una Vivienda



$$\begin{matrix} x & & y \\ M^2 * a + b & = & \text{Precio} \end{matrix}$$

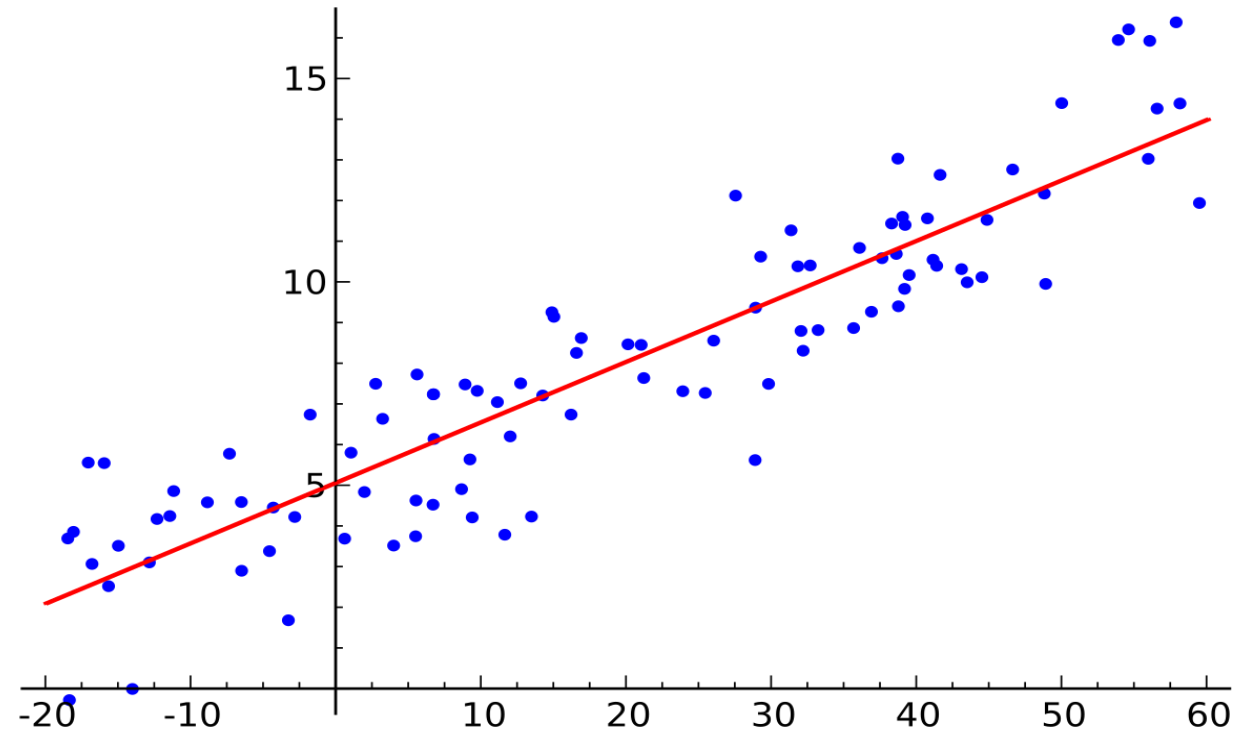
$$M^2 \longrightarrow \text{Precio}$$

Regresión lineal

Se utiliza principalmente para Modelos de **Regresión**

Donde la Variable objetivo es **Continua** (numero real)

Se busca una dependencia de este objetivo en relación a las otras variables



Regresión lineal

sklearn.linear_model.LinearRegression

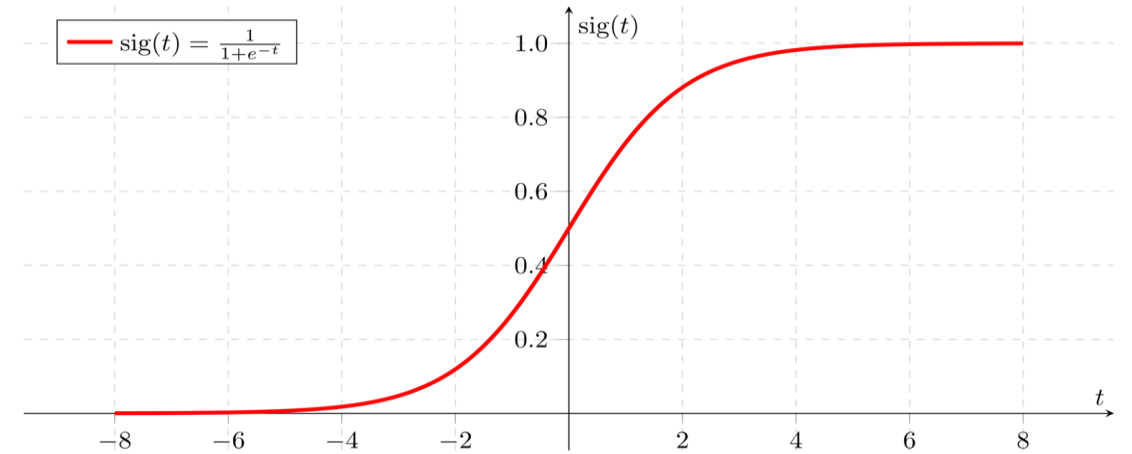
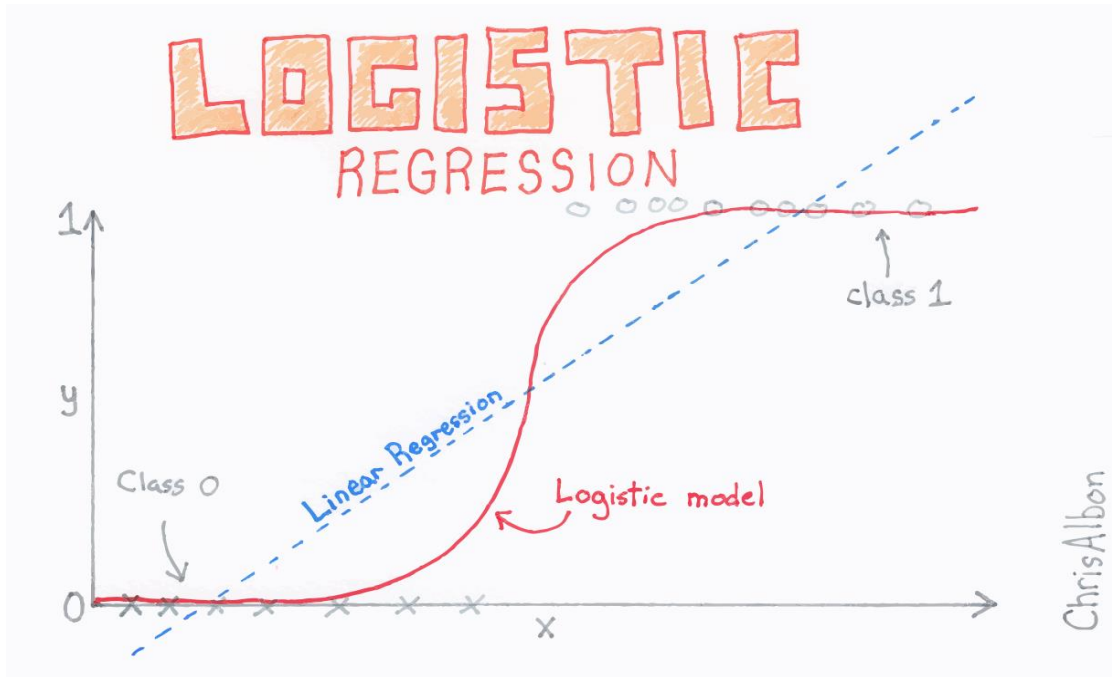
```
class sklearn.linear_model. LinearRegression (fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)
```

[\[source\]](#)

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0000...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

>>>

Regresión Logística



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

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

Regresión Logística

`sklearn.linear_model.LogisticRegression`

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)
```

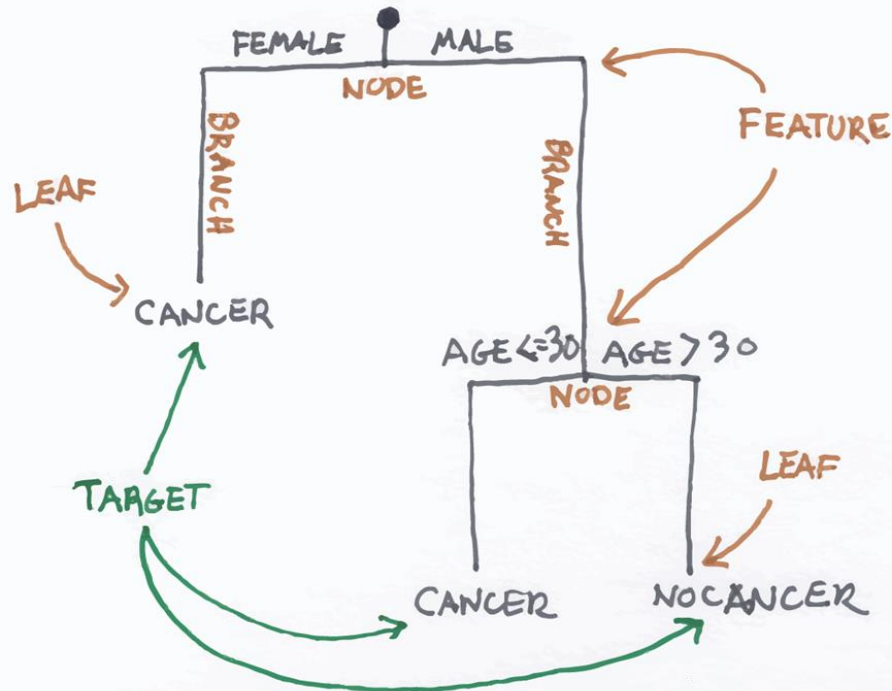
[\[source\]](#)

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Arbol de Decisión

Arboles de Decisión

DECISION TREES



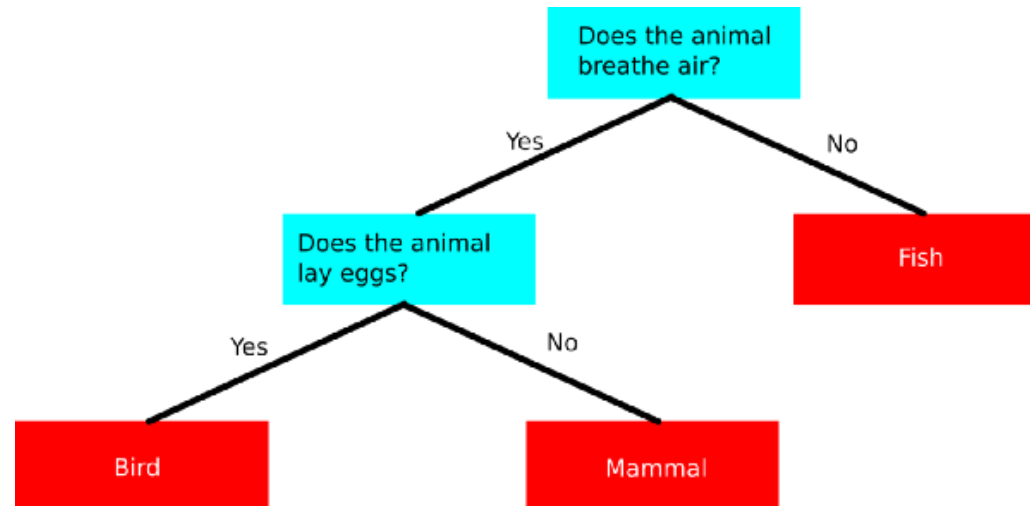
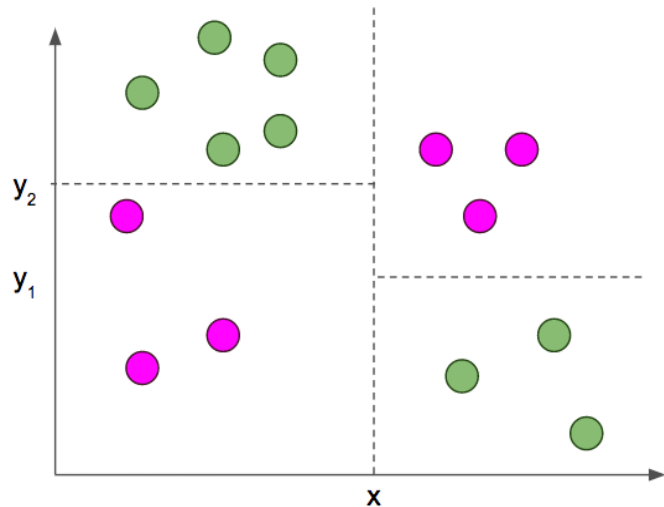
DECISION TREES HAVE HIGH INTERPRETABILITY. AFTER BEING TRAINED YOU CAN LITERALLY DRAW THEM.

SPLITS DATA ON THE FEATURE WHICH WHEN SPLIT PROVIDES HIGHEST INFORMATION GAIN.

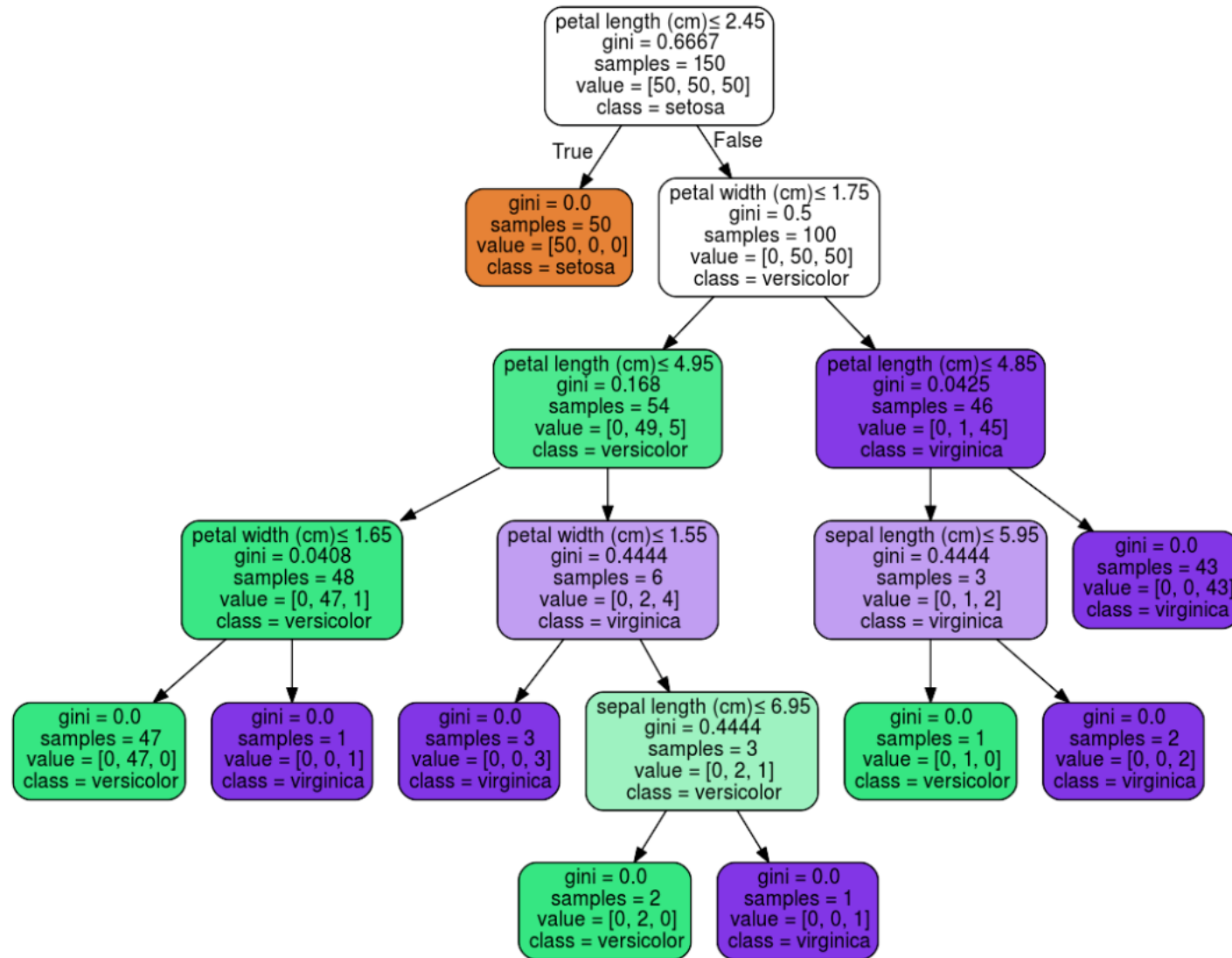
BY CHRIS ALBON

Arboles de Decisión

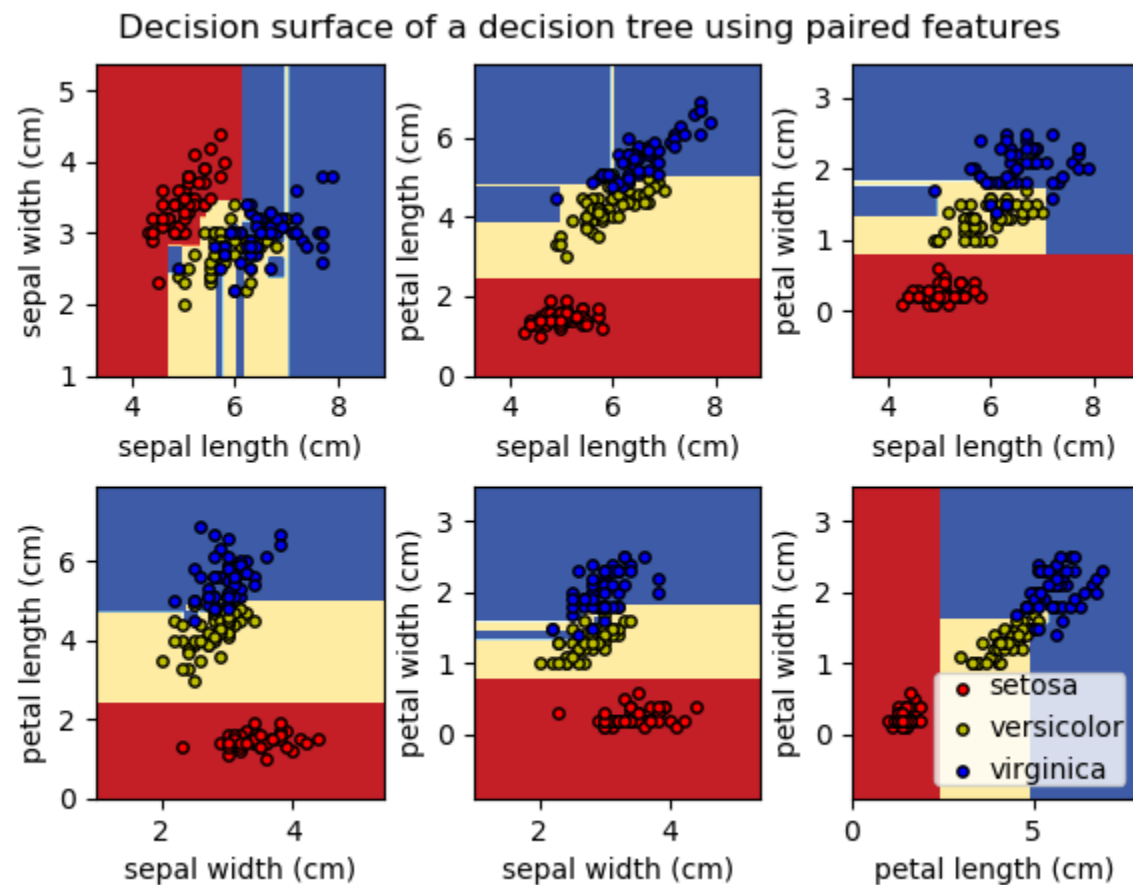
En función de los datos de entrenamiento, el modelo construye los niveles jerárquicos y las preguntas a realizar. Estas preguntas son determinadas en función de las “features” de nuestro dataset. El modelo de decision trees classification soporta problemas “multi-clase”.



Arboles de Decisión - Iris



Arboles de Decisión - Iris



Arboles de Decisión

`sklearn.tree`.DecisionTreeClassifier

```
class sklearn.tree. DecisionTreeClassifier (criterion='gini', splitter='best', max_depth=None,  
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,  
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

[\[source\]](#)

```
>>> from sklearn import tree  
>>> X = [[0, 0], [1, 1]]  
>>> Y = [0, 1]  
>>> clf = tree.DecisionTreeClassifier()  
>>> clf = clf.fit(X, Y)
```

Vecinos más Cercanos (KNN)

KNN – K Vecinos más Cercanos

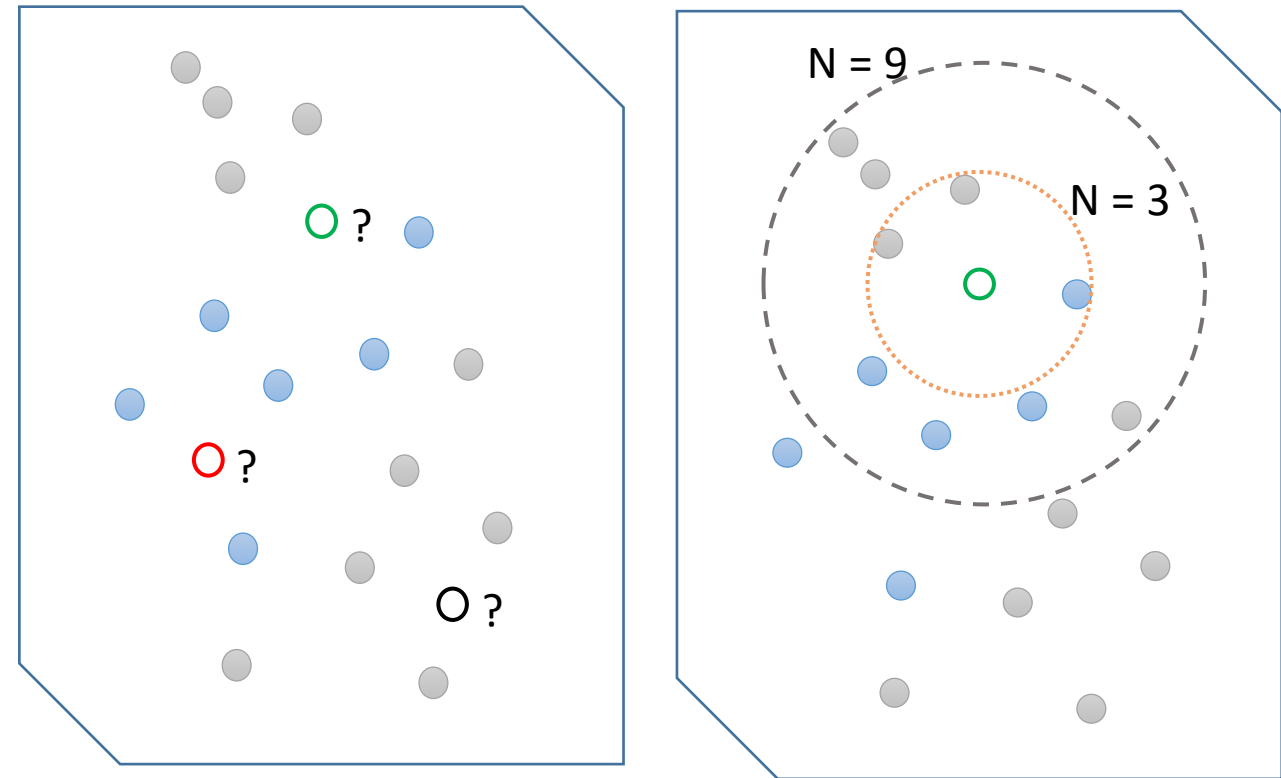
Requiere datos numéricos

Los datos deben estar normalizados

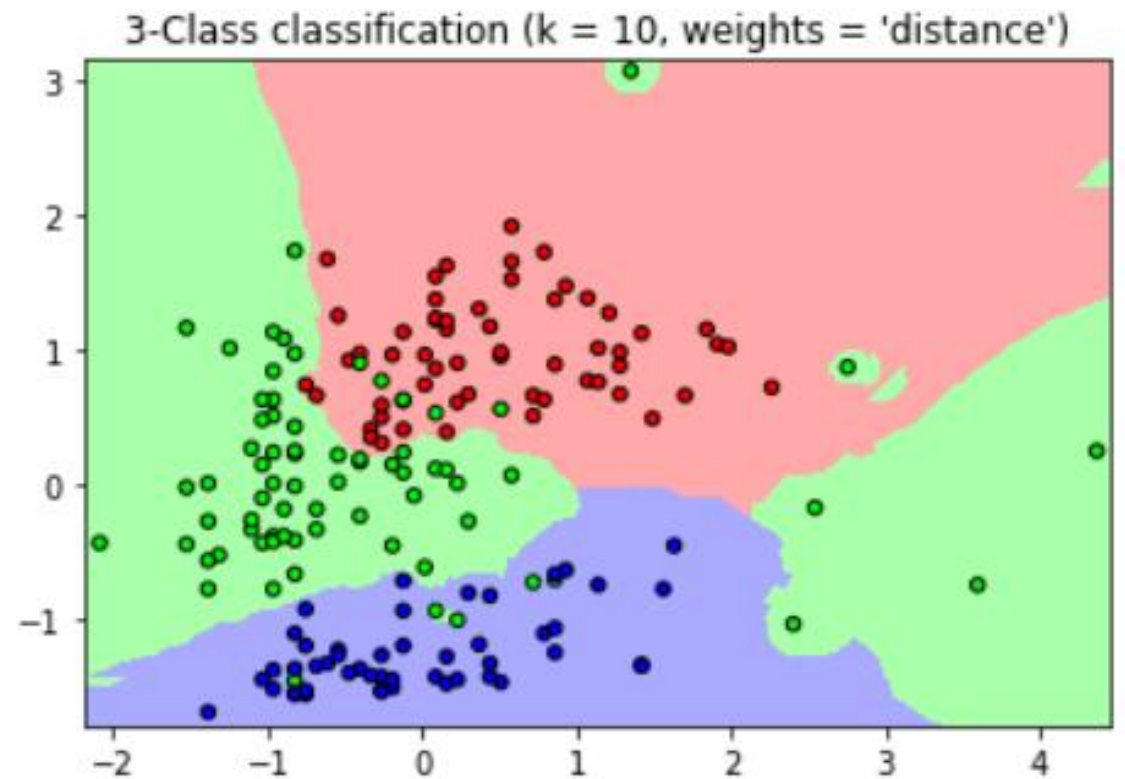
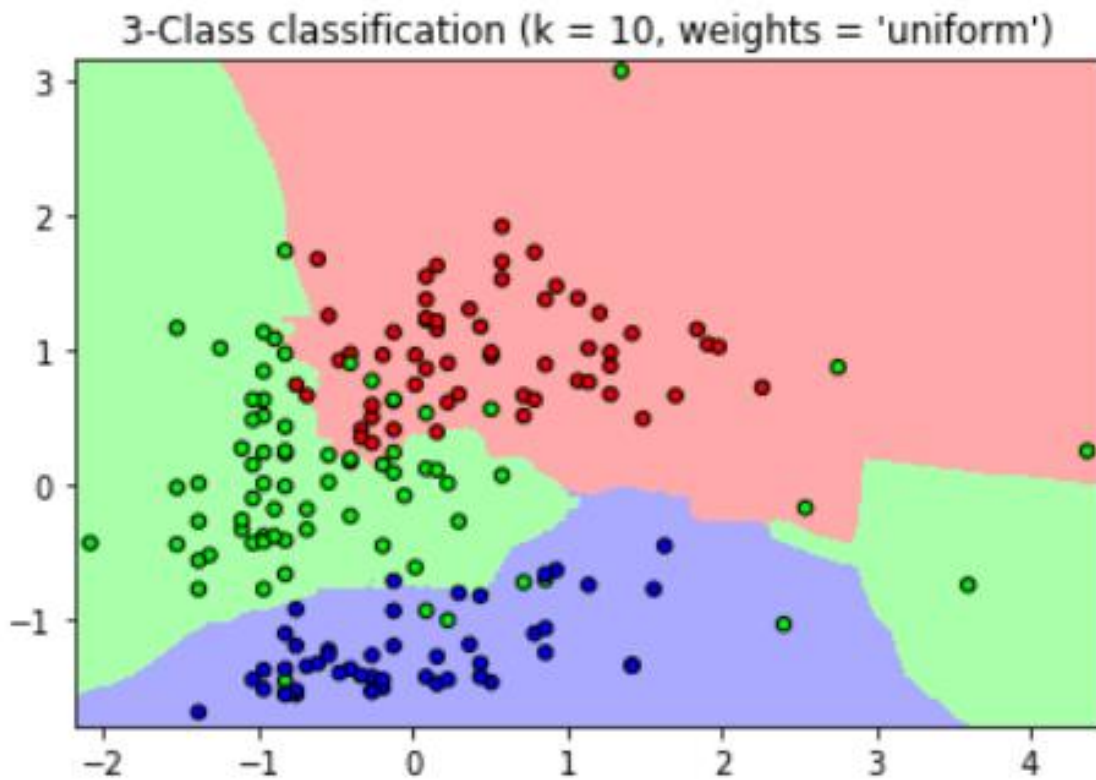
Medida de distancia (Euclidean)

Se define la Cantidad de Vecinos

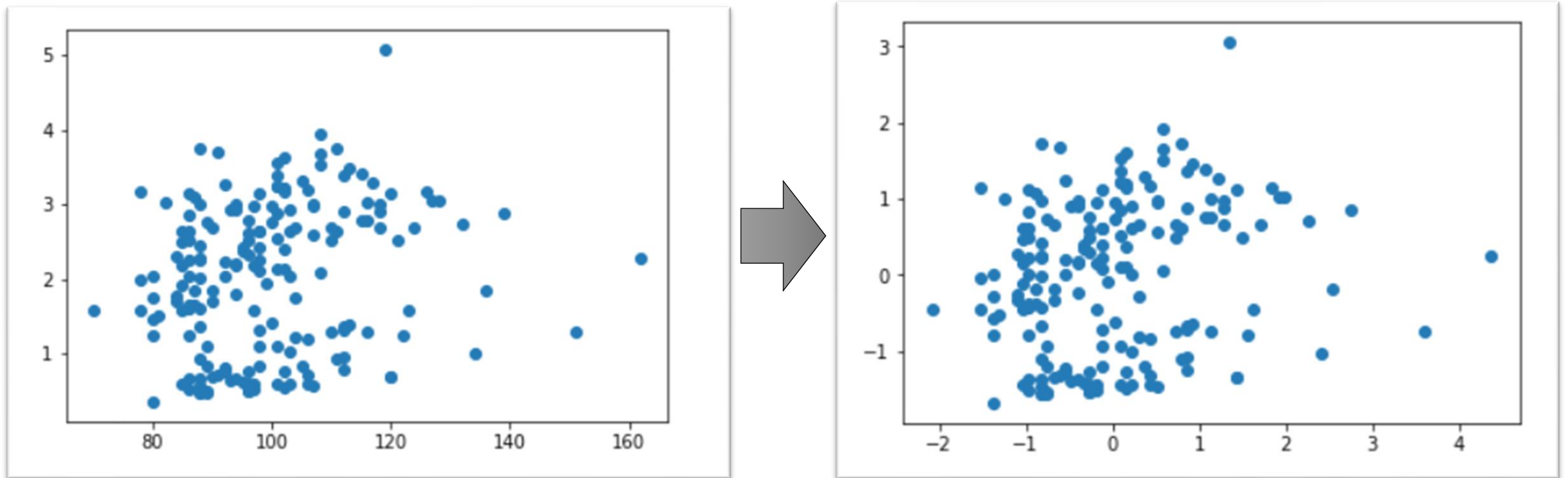
Se pueden dar “pesos” a los puntos



KNN – K Vecinos más Cercanos



KNN – K Vecinos más Cercanos



KNN – K Vecinos más Cercanos

`sklearn.neighbors.KNeighborsClassifier` ¶

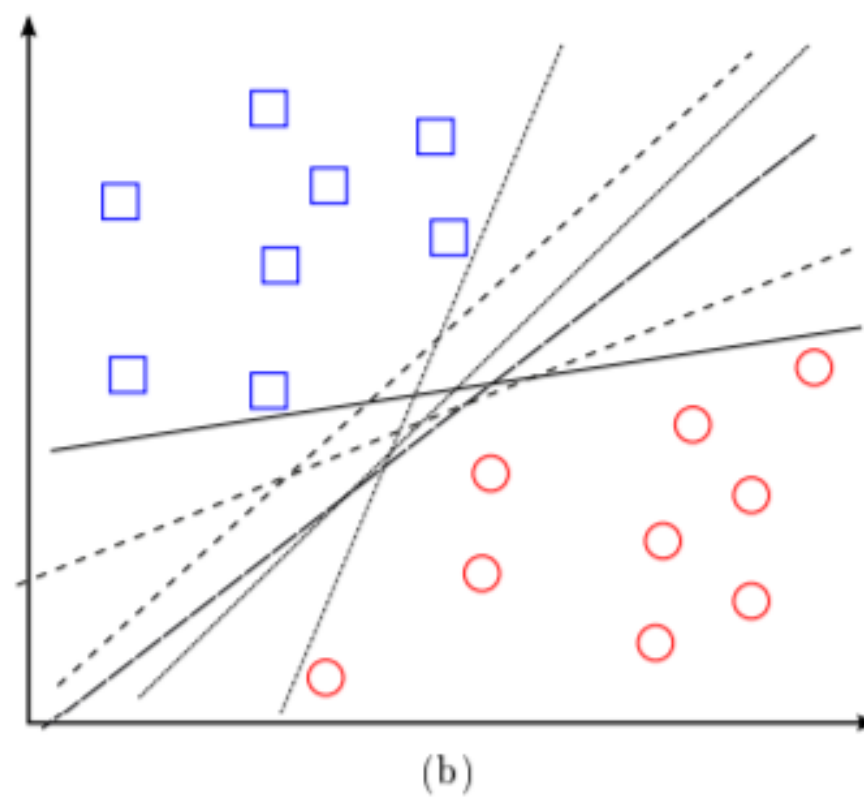
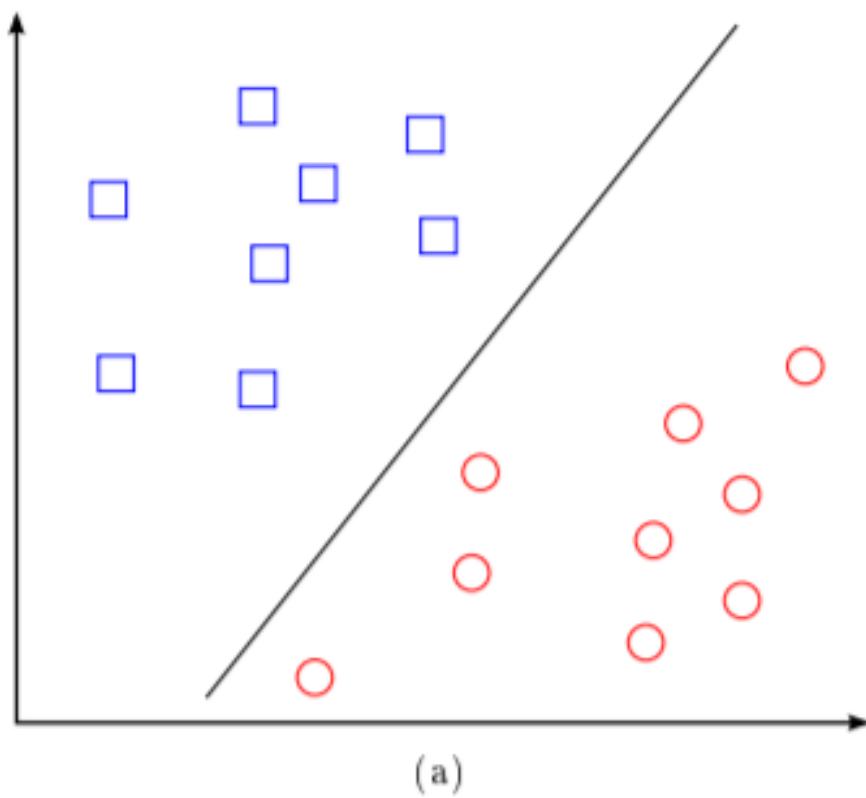
```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,  
p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

[\[source\]](#)

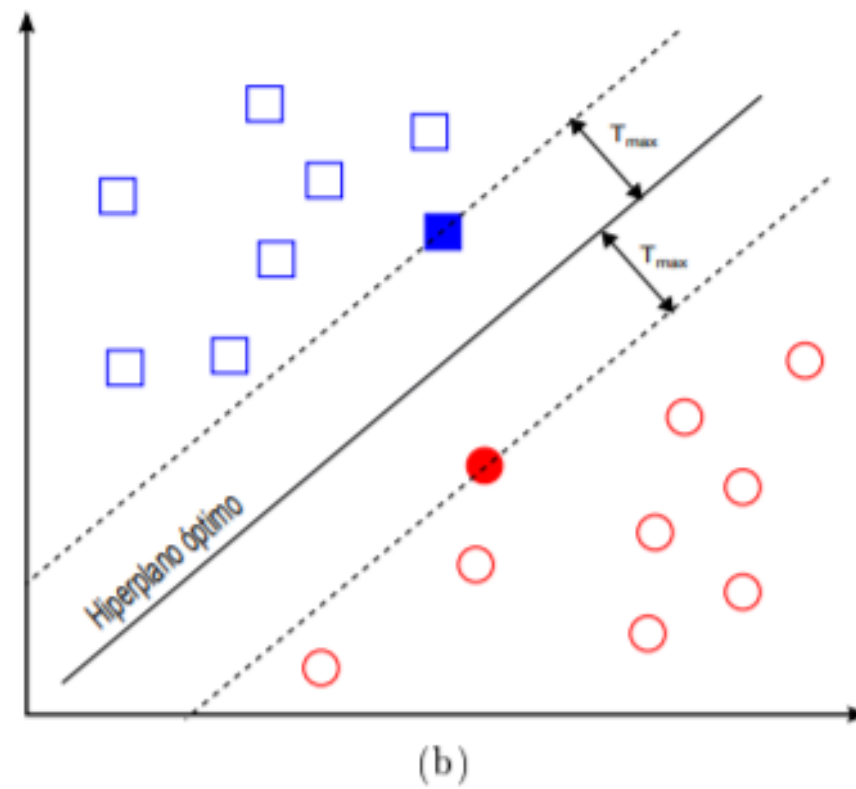
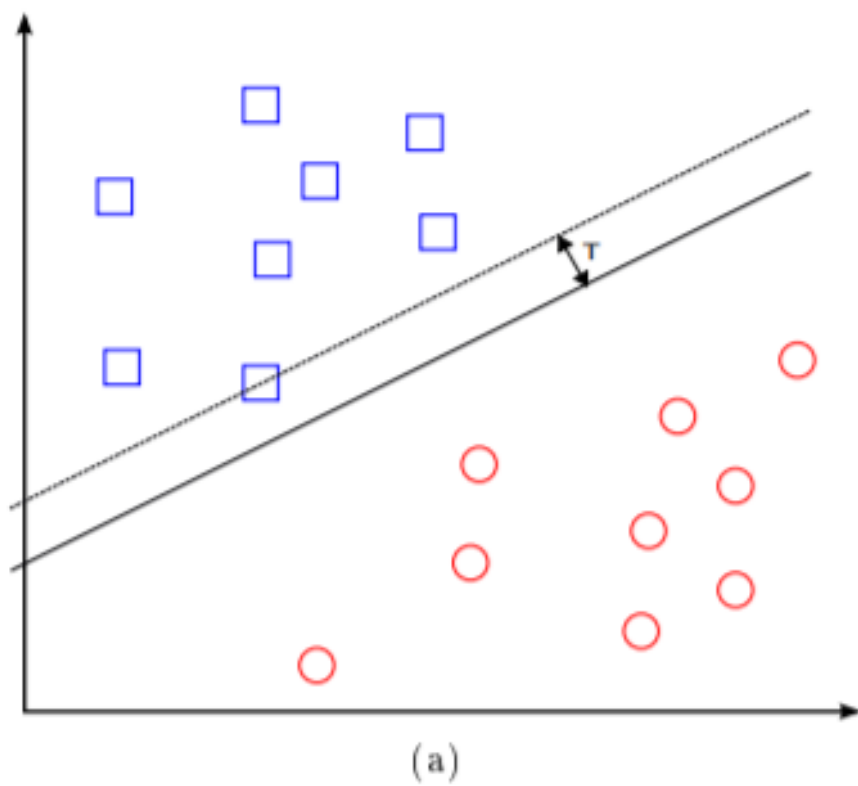
```
>>> X = [[0], [1], [2], [3]]  
>>> y = [0, 0, 1, 1]  
>>> from sklearn.neighbors import KNeighborsClassifier  
>>> neigh = KNeighborsClassifier(n_neighbors=3)  
>>> neigh.fit(X, y)  
KNeighborsClassifier(...)  
>>> print(neigh.predict([[1.1]]))  
[0]  
>>> print(neigh.predict_proba([[0.9]]))  
[[0.66666667 0.33333333]]
```


Support Vector Machine

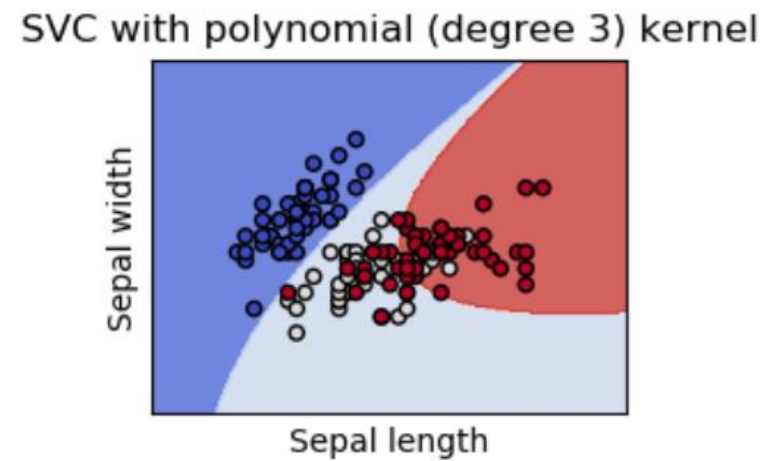
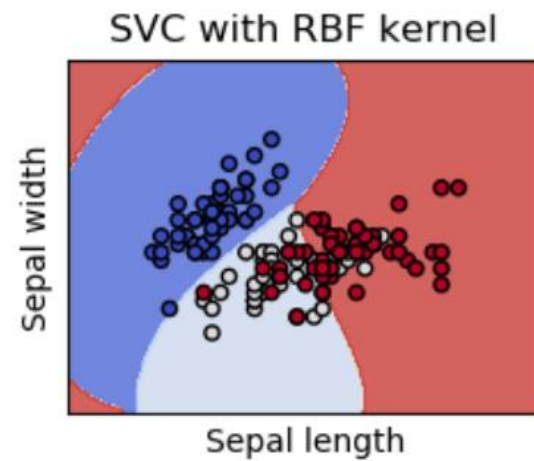
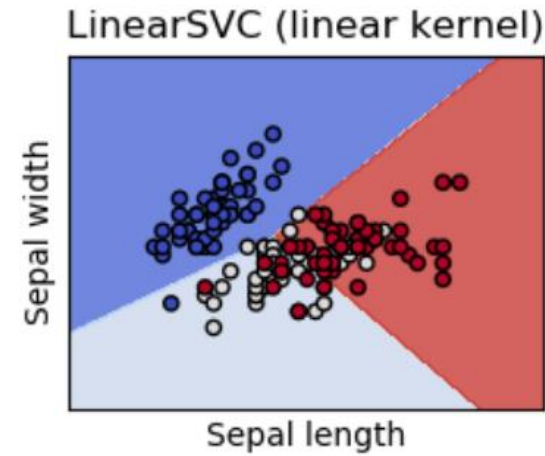
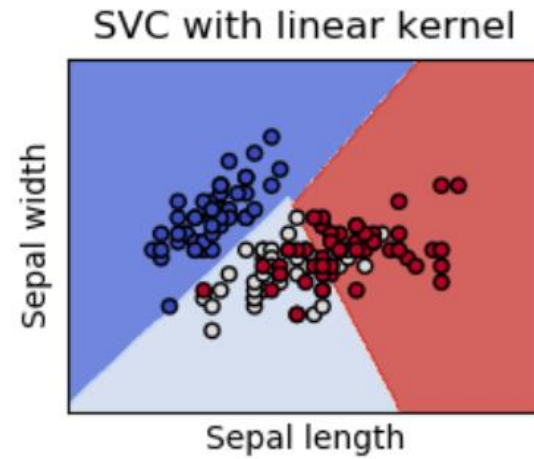
Support Vector Machine



Support Vector Machine



Support Vector Machine



Support Vector Machine

`sklearn.svm.SVC`

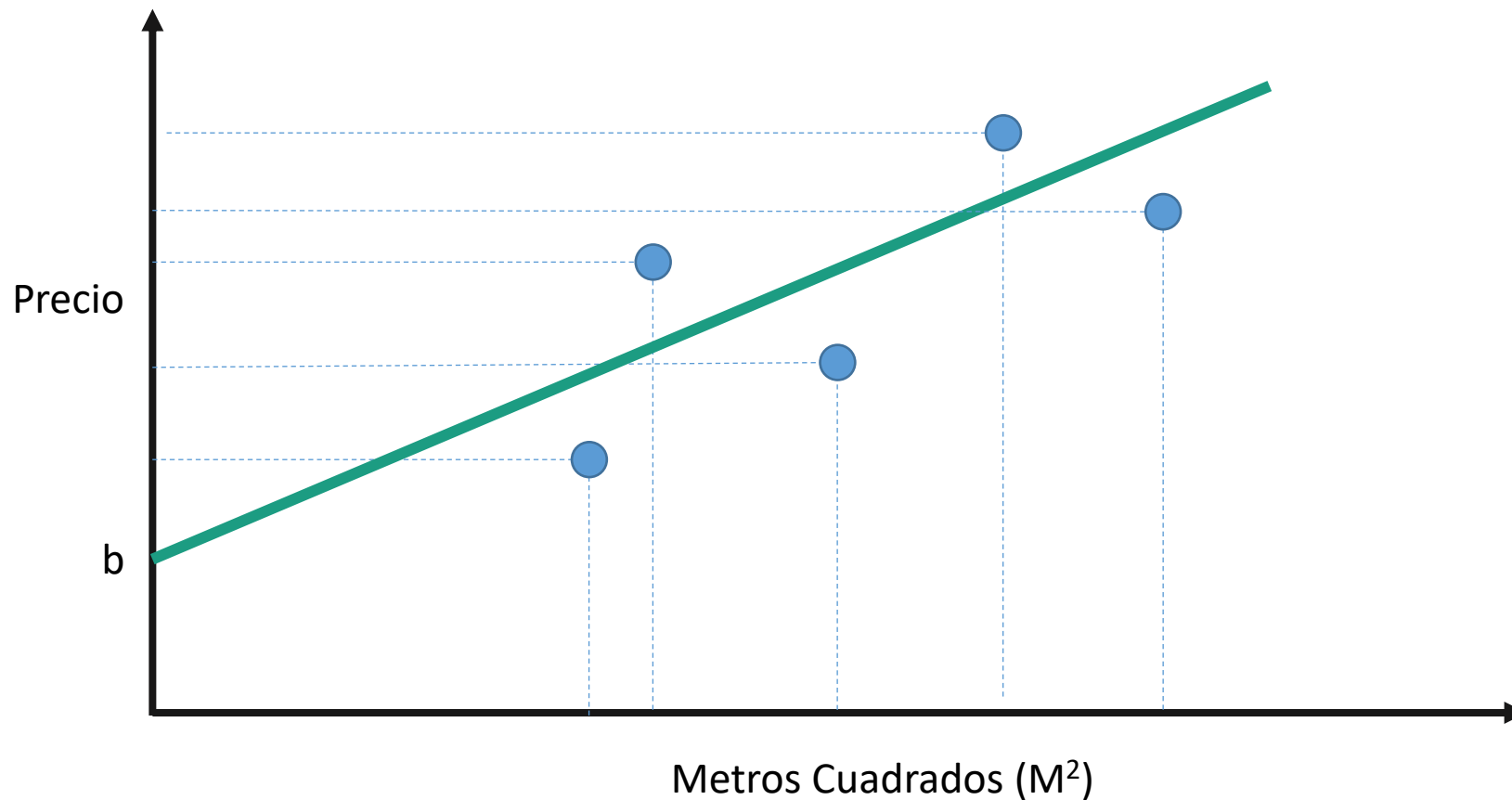
```
class sklearn.svm. SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
```

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC(gamma='scale')
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Modelos Avanzados

Redes Neuronales

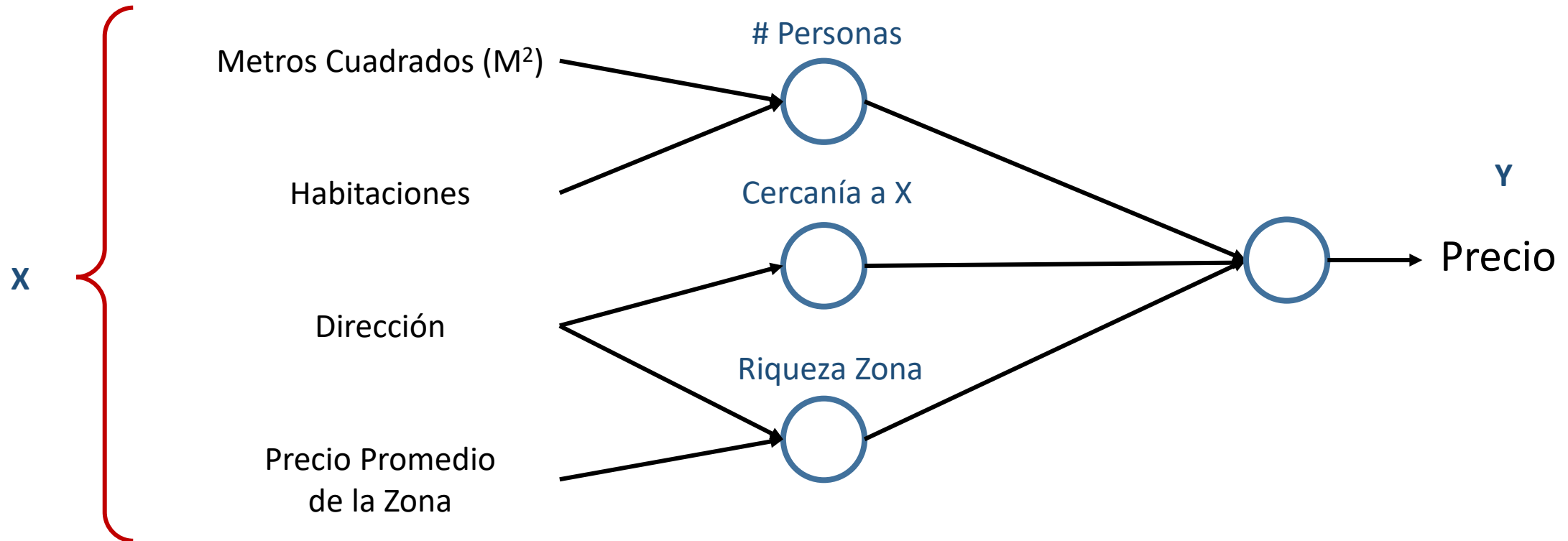
Predecir el precio de una Vivienda



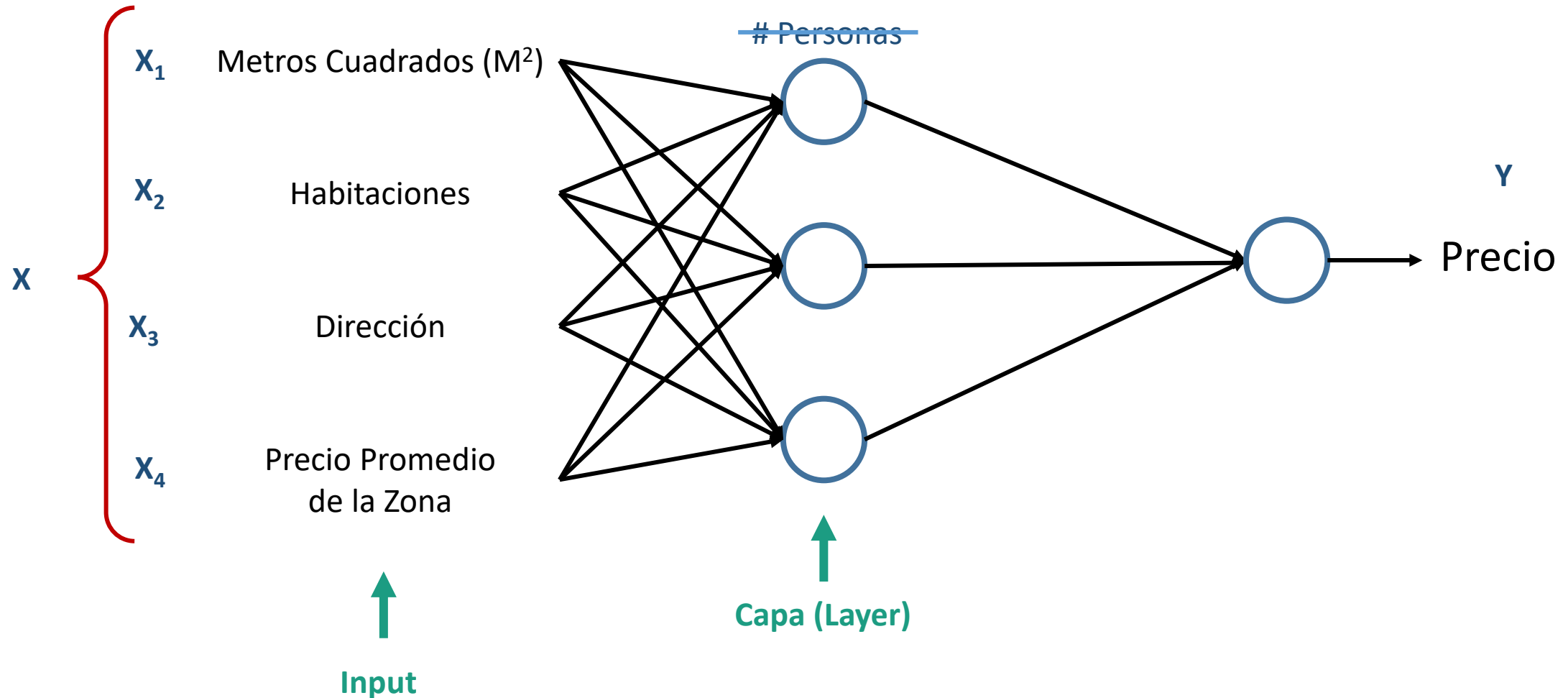
$$\begin{array}{c} x \qquad \qquad \qquad y \\ M^2 * a + b = \text{Precio} \\ \underbrace{\hspace{1.5cm}} \end{array}$$

$M^2 \rightarrow \text{Neurona} \rightarrow \text{Precio}$

Predecir el precio de una Vivienda

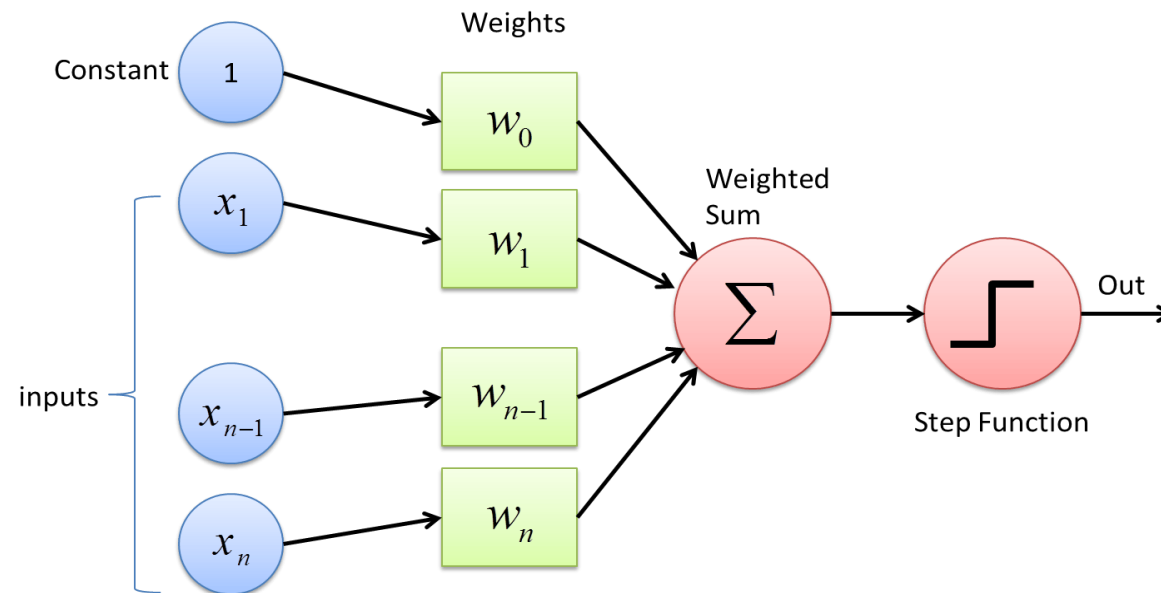


Predecir el precio de una Vivienda



Perceptrón

Es un algoritmo de Clasificación Lineal originado en los años 50s. Por cada “feature” tendremos una entrada al modelo y cada entrada se le asignará un “peso”. Una vez asignados los pesos se deberá sumar todas las entradas con sus pesos y determinar el valor de salida **$(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$** . El resultado se determina por la función de activación, que en el caso de ser binario resultará como 0 o 1 (en otros casos -1 o 1).



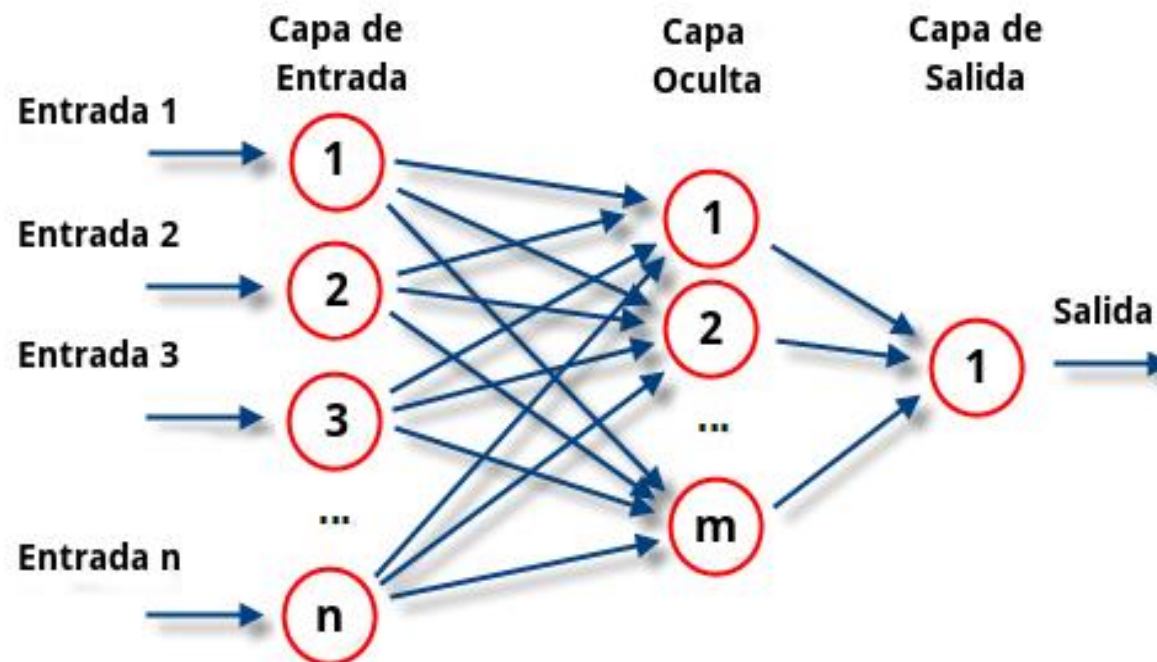
Perceptrón

- Es un modelo que trabaja “online”. Procesa una muestra a la vez. De esta manera los pesos también se actualizan gradualmente.
- Únicamente en los casos donde el modelo realiza una predicción errónea los pesos se actualizan.
- El perceptrón computa una función de costo basada en la cantidad de veces que hubo una mal clasificación.
- Puede utilizar distintas funciones de activación:
 - Signo \rightarrow es -1 si $x < 0$ y $+1$ si $x > 0$.
 - Sigmoidea \rightarrow ya vista en la clase de regresión logística
 - TanH
 - ReLu

Perceptrón Multicapa (Multilayer Perceptron)

Si al modelo perceptrón le incorporamos capas intermedias entre la entrada y la salida obtenemos lo que comúnmente se llama “red neuronal con capas ocultas”.

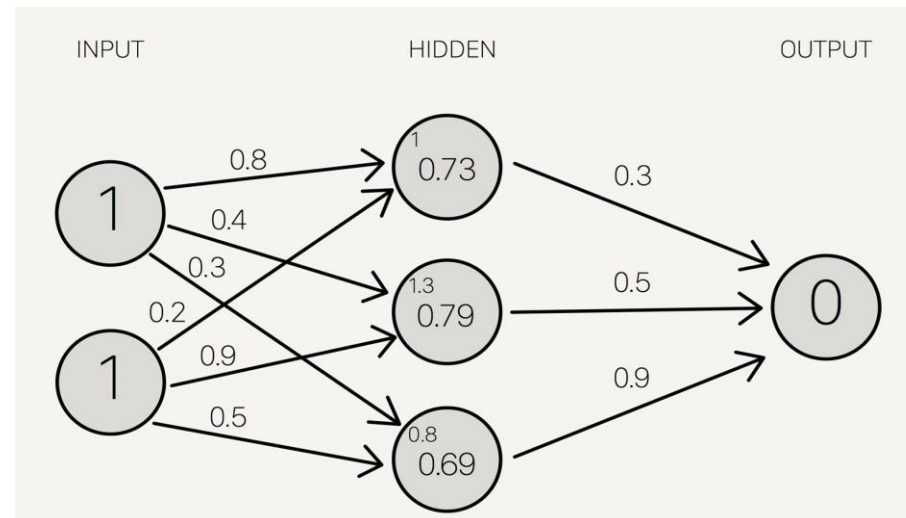
El hecho de tener capas ocultas en el medio implica procesar la entrada con otro perceptrón lineal. Esta arquitectura permite poder captar patrones **no lineales**.



Perceptrón Multicapa (Multilayer Perceptron)

En el entrenamiento se debe determinar el valor de los pesos (cuánto “pesa” cada flecha), tanto de la primer capa, como de las capas intermedias.

La herramienta para calcular los pesos se denomina “**backpropagation**” y consiste en calcular los gradientes de cada capa a medida que pasan muestras por la red y son evaluadas.



Ejemplo Visual: <https://playground.tensorflow.org>

Perceptrón Multicapa (Multilayer Perceptron)

`sklearn.neural_network.MLPClassifier`

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

[\[source\]](#)

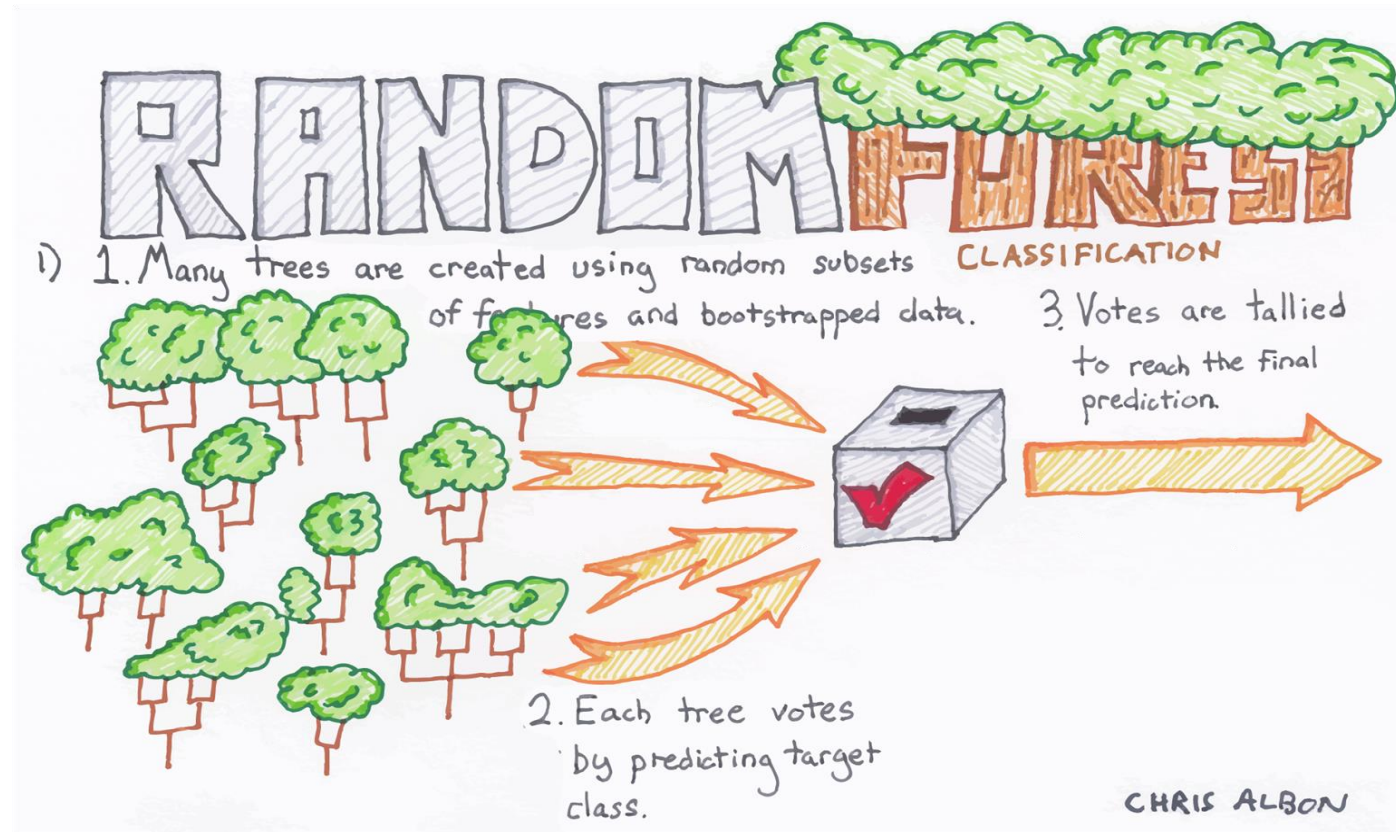
```
>>> from sklearn.neural_network import MLPClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
...                                                    random_state=1)
>>> clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
>>> clf.predict_proba(X_test[:1])
array([[0.038..., 0.961...]])
>>> clf.predict(X_test[:5, :])
array([1, 0, 1, 0, 1])
>>> clf.score(X_test, y_test)
0.8...
```

>>>

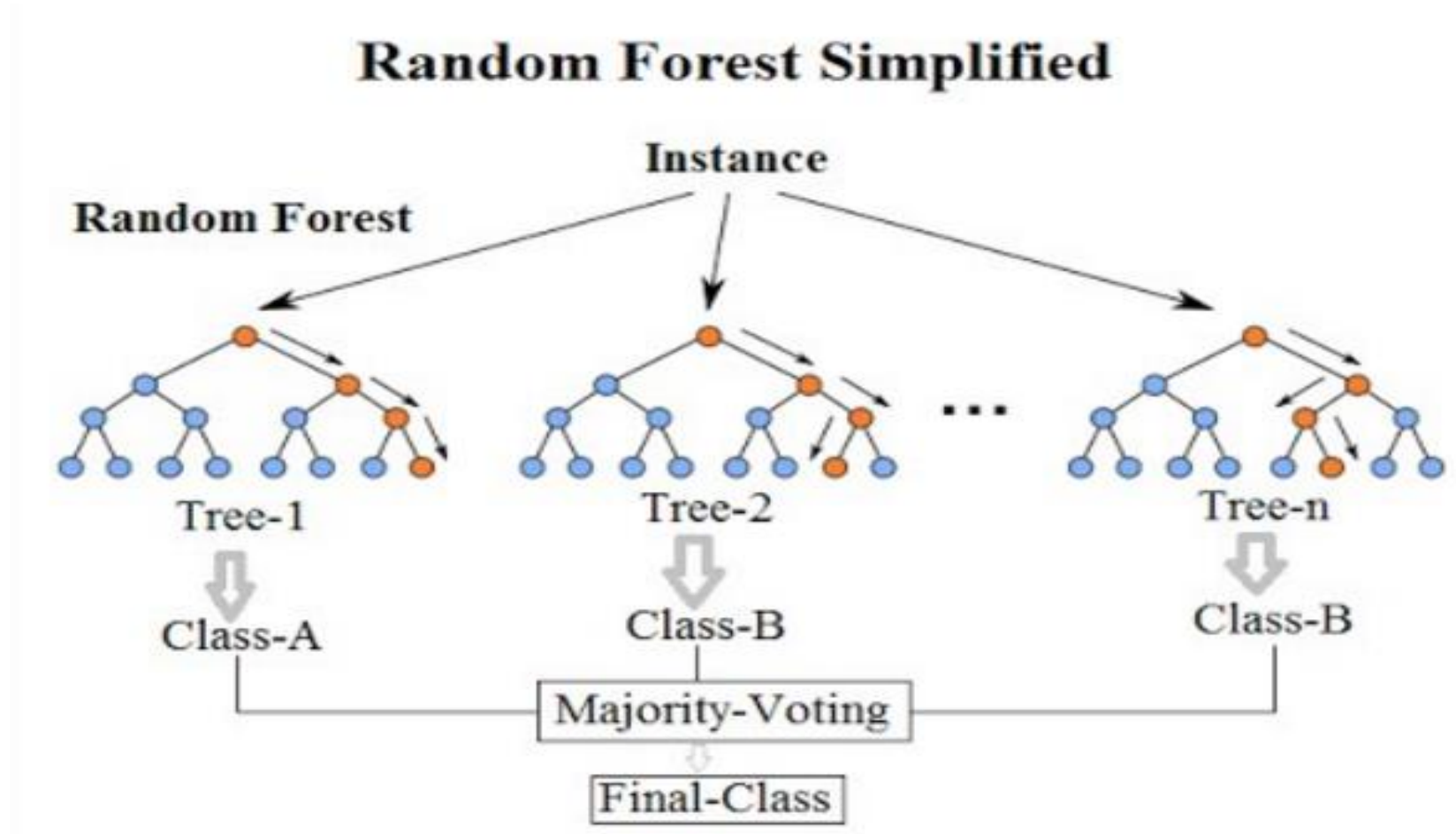
Bagging

Random Forest

Consiste en un conjunto de varios modelos de Decision Trees, todos realizan una predicción sobre un mismo dataset. La respuesta final contempla el resultado de todos los modelos, generando una respuesta mucho más robusta que un solo clasificador.



Random Forest



Random Forest

3.2.4.3.1. `sklearn.ensemble.RandomForestClassifier`

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

[\[source\]](#)

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

Boosting

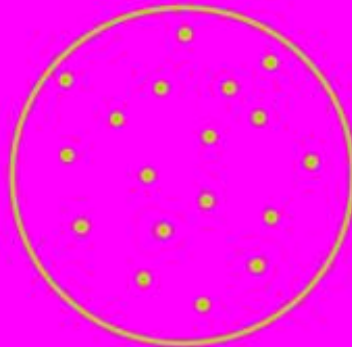
En qué consiste el Boosting?

El boosting consiste en combinar los resultados de varios clasificadores débiles para obtener un clasificador robusto.

Cuando se añaden estos clasificadores débiles, se lo hace de modo que estos tengan diferente peso en función de la exactitud de sus predicciones.

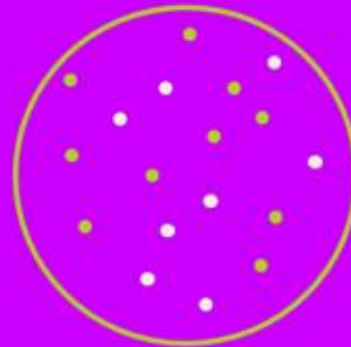
Luego de que se añade un clasificador débil, los datos cambian su estructura de pesos: los casos que son mal clasificados ganan peso y los que son clasificados correctamente pierden peso.

single



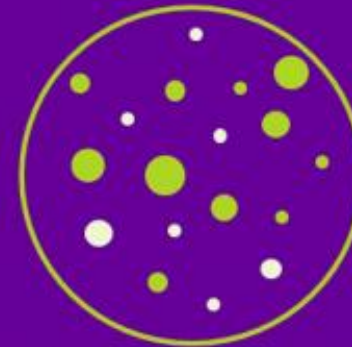
complete training set

bagging



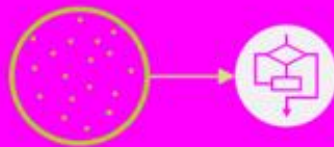
random sampling with
replacement

boosting



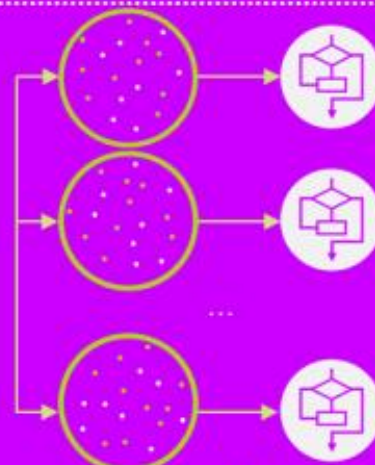
random sampling with
replacement
over weighted data

single



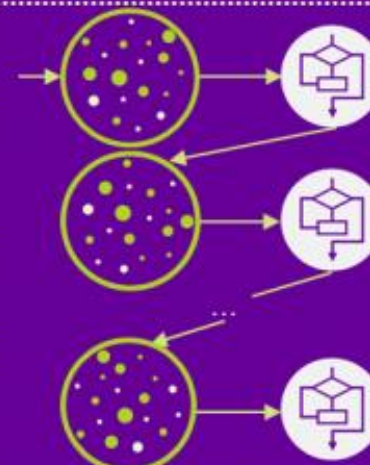
1 iteration

bagging



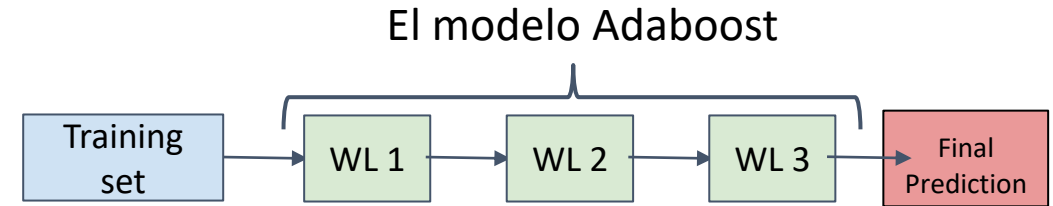
parallel

boosting



sequential

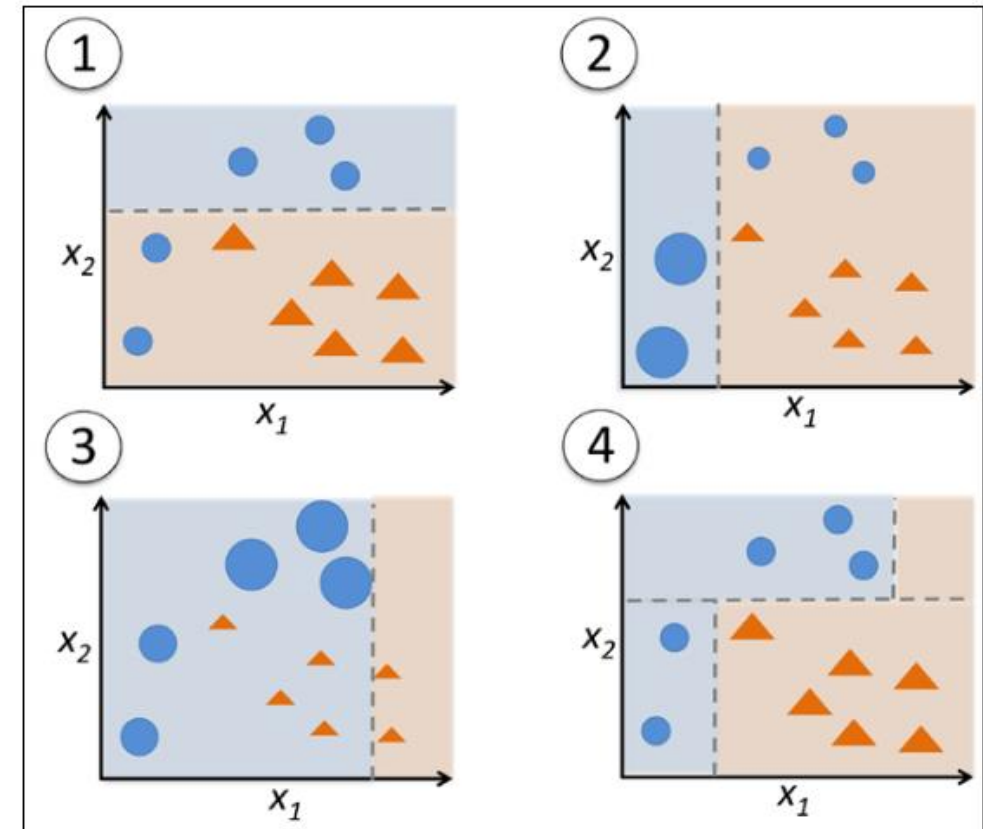
Adaboost



Construcción del algoritmo:

1. Elegir un “weak-learner”(árboles de bajo nivel)
2. Entrenarlo con todo el Training set
3. Identificar en qué instancias de entrenamiento la clasificación falla
4. Aprender de los errores: entrenar el siguiente “weak-learner” haciendo énfasis en las instancia de entrenamiento mal clasificadas
5. Repetir hasta conseguir el score deseado.

Modelo final: secuencia de “weak-learners”.



Adaboost

`sklearn.ensemble.AdaBoostClassifier`

```
class sklearn.ensemble.AdaBoostClassifier(base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R',  
random_state=None) ⓘ
```

[\[source\]](#)

```
>>> from sklearn.ensemble import AdaBoostClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> clf = AdaBoostClassifier(n_estimators=100, random_state=0)
>>> clf.fit(X, y)
AdaBoostClassifier(n_estimators=100, random_state=0)
>>> clf.predict([[0, 0, 0, 0]])
array([1])
>>> clf.score(X, y)
0.983...
```

>>>

Gradient Boosting

Descenso del Gradiente

La idea de la potenciación del gradiente puede ser interpretado como un algoritmo de optimización en una función de coste adecuada.

Es decir, algoritmos que optimizan una función de coste sobre el espacio de función mediante la elección iterativa de una función (hipótesis débil) que apunta en la dirección del gradiente negativo.

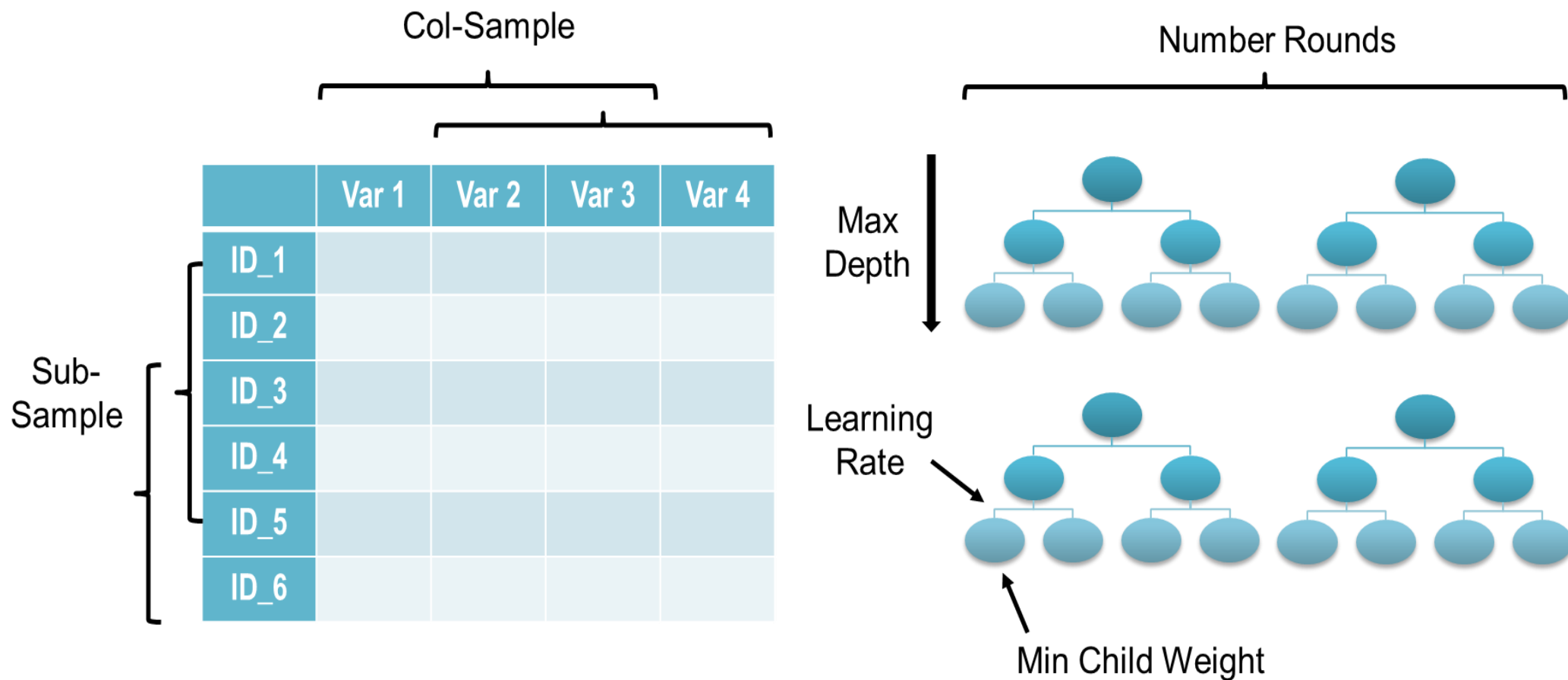
Implementaciones Famosas:

XGBoost (DMLC)

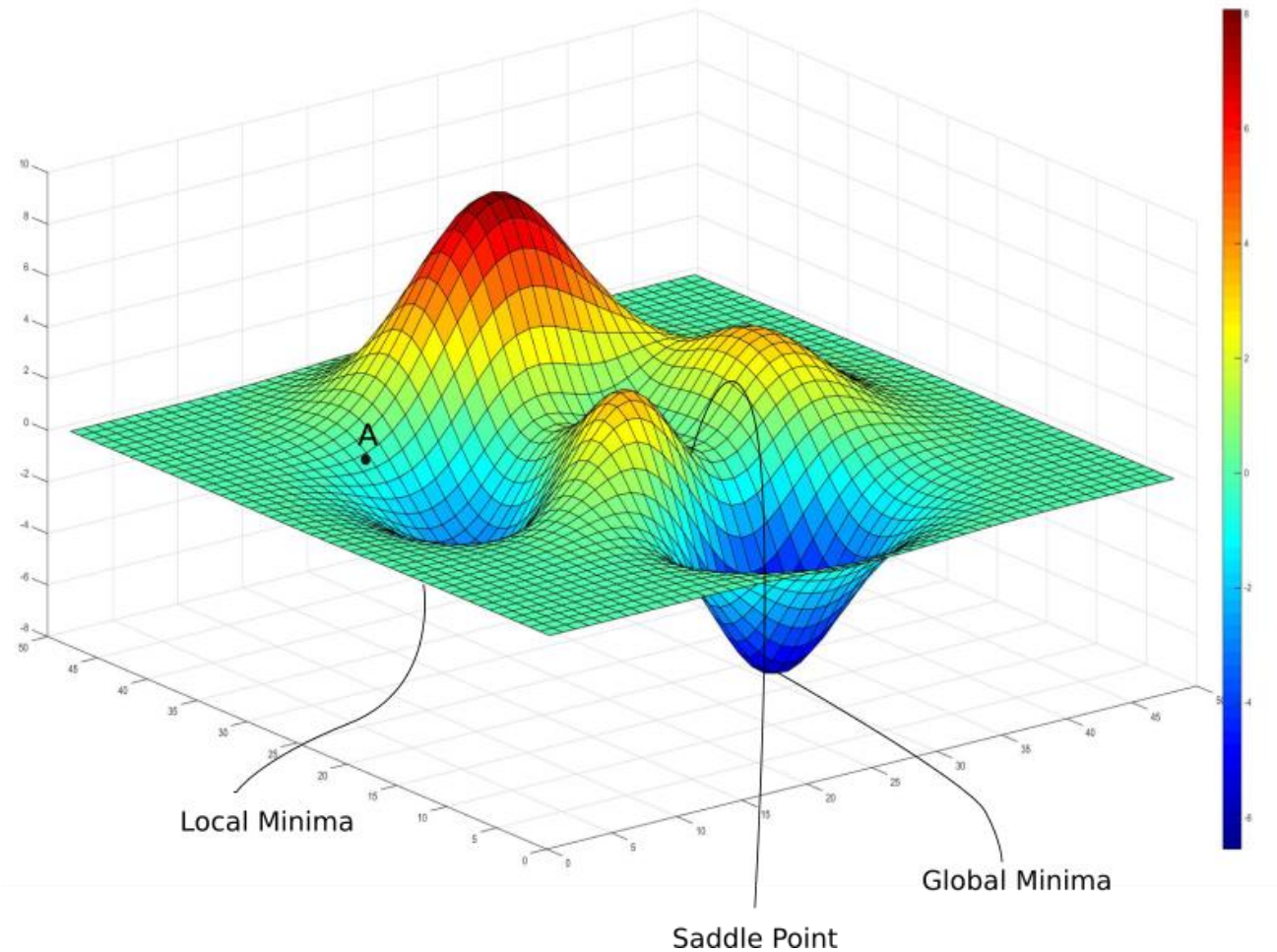
LightGBM (Microsoft)

Ejemplo Visual: http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html

XGBoost – Definición de Parámetros



Graficar el Descenso del Gradiente



Gradient Boosting

`sklearn.ensemble.HistGradientBoostingClassifier`

```
class sklearn.ensemble.HistGradientBoostingClassifier(loss='auto', *, learning_rate=0.1, max_iter=100, max_leaf_nodes=31,
max_depth=None, min_samples_leaf=20, l2_regularization=0.0, max_bins=255, monotonic_cst=None, warm_start=False,
early_stopping='auto', scoring='loss', validation_fraction=0.1, n_iter_no_change=10, tol=1e-07, verbose=0, random_state=None) ↑
```

[\[source\]](#)

```
>>> # To use this experimental feature, we need to explicitly ask for it:
>>> from sklearn.experimental import enable_hist_gradient_boosting # noqa
>>> from sklearn.ensemble import HistGradientBoostingClassifier
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> clf = HistGradientBoostingClassifier().fit(X, y)
>>> clf.score(X, y)
1.0
```

>>>

Ensembles de Modelos

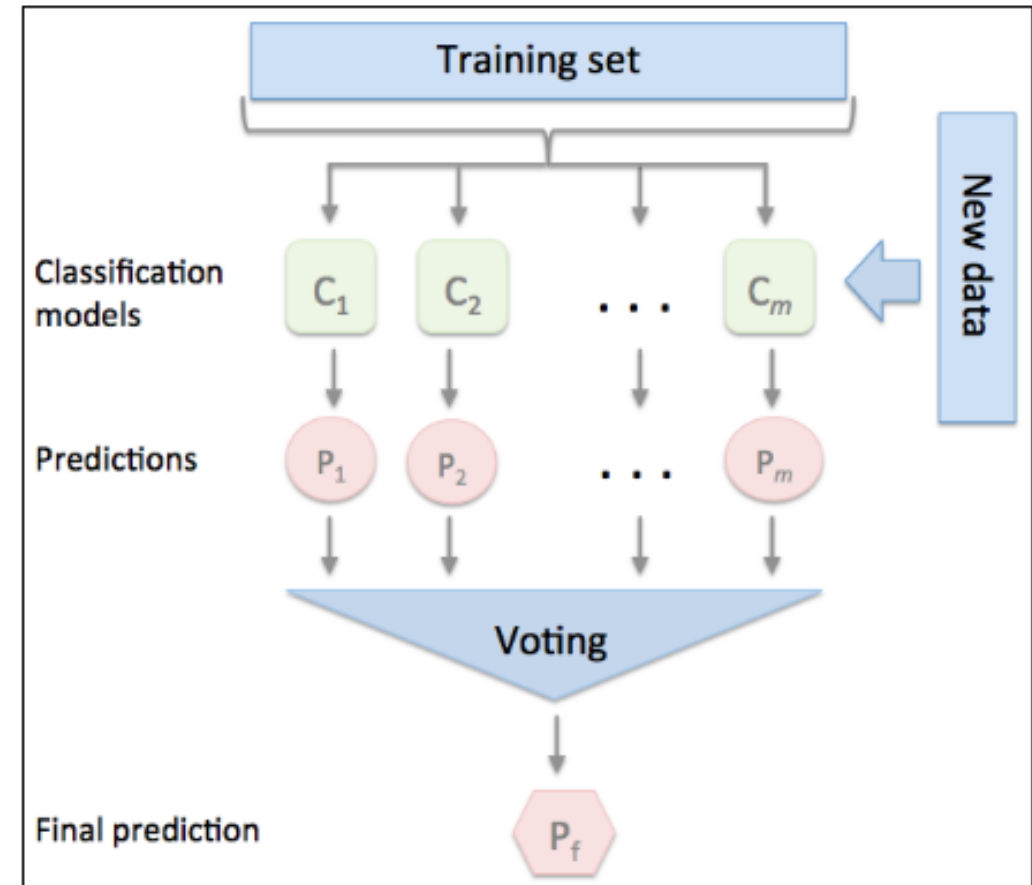
Stacking

Stacking de modelos por votación

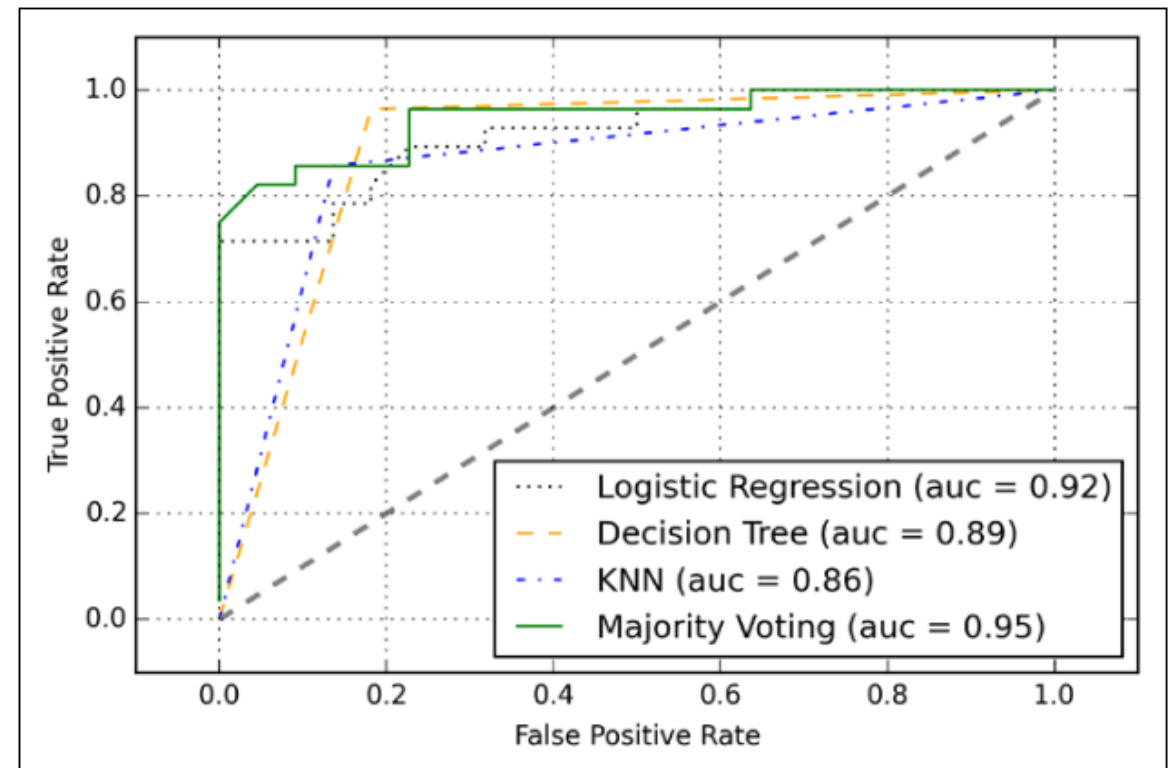
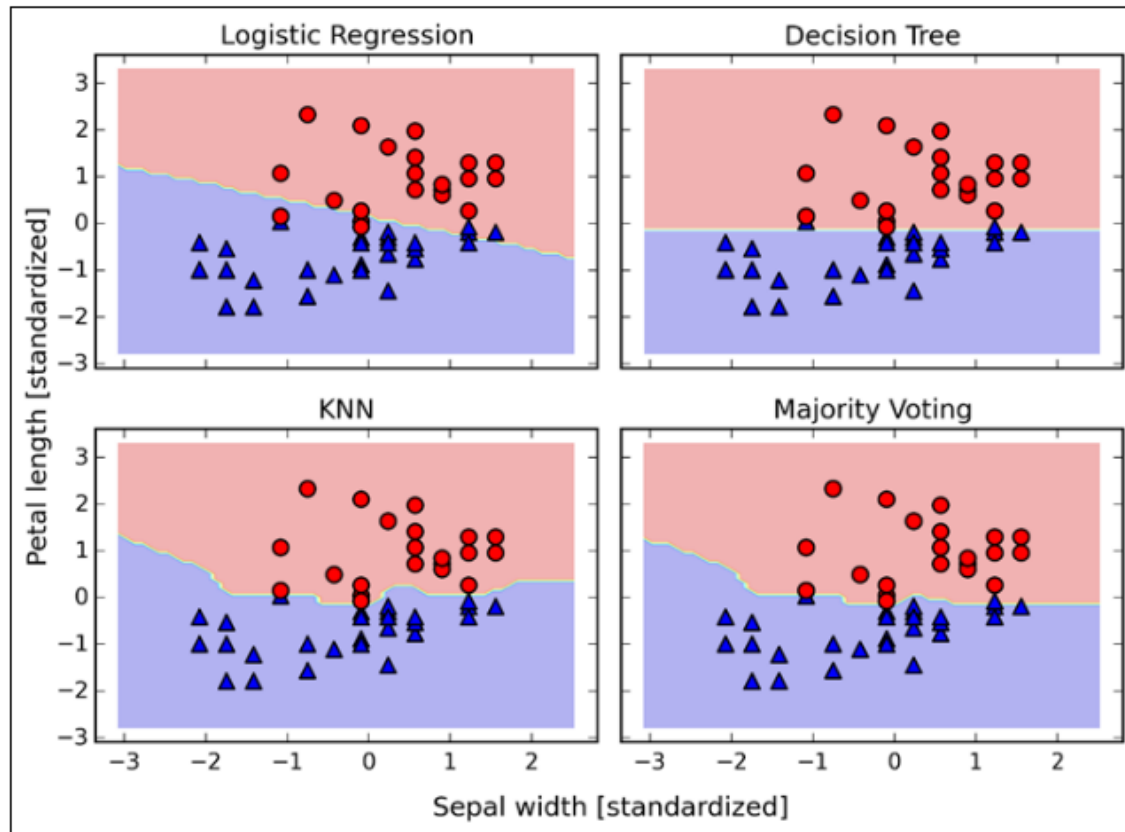
Construcción del algoritmo:

1. Elegir el número de clasificadores a combinar
2. Separar el Training set en m subconjuntos
3. Entrenar los m clasificadores, cada uno con un subconjunto del Training set

Predicción del ensemble: votación por mayoría



Ensemble de modelos por votación





UCEMA

Muchas Gracias!