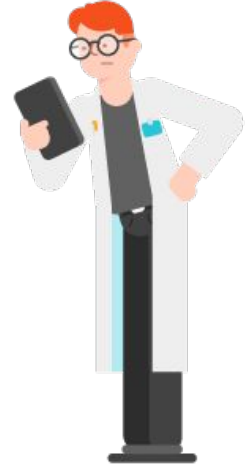
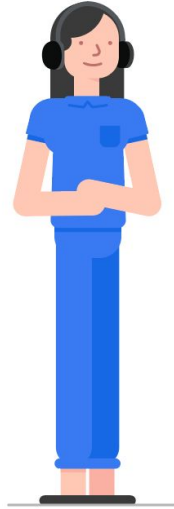
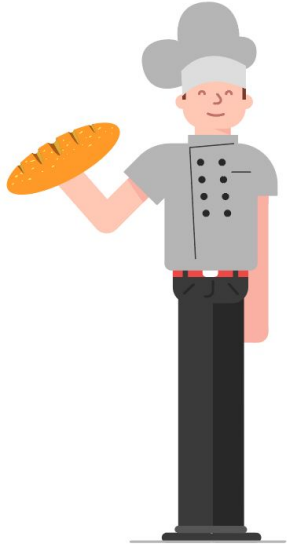


Supervised Learning

Sebastien Perez

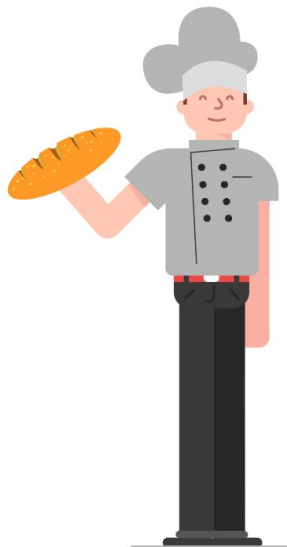
El Machine Learning es una oportunidad

#GoogleActívale



Panadero/a

#GoogleActívale



El pan es un producto altamente perecedero y requiere producirlo en cantidad adecuada.

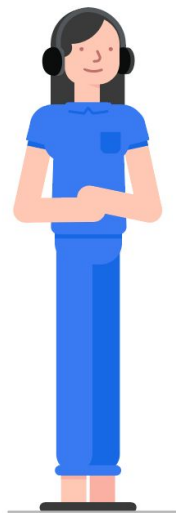
■ **Desafío:** pérdida de producto por invendidos.

■ **Datos:**

- Ventas pasadas
- Indicadores económicos de la zona
- Calendario de eventos (Vacaciones)

Operario/a de fábrica

#GoogleActívate



Debe velar por el buen funcionamiento de sus máquinas y evitar paradas.

- **Desafío:** la parada de una máquina produce pérdidas para la fábrica.
- **Datos:**
 - Histórico de medidas de sensores en la máquina en distintos puntos
 - Datos de los tests de uso del vendedor

Responsable de Marketing

#GoogleActívate



Monitoriza el estado del negocio y vela por maximizar los ingresos.

- **Desafío:** maximizar la tasa de conversión
- **Datos:**
 - Compras de los clientes en el pasado
 - Preferencias
 - Indicadores económicos de la zona

Médico

#GoogleActívale



Analiza radiografías para detectar alguna enfermedad.

- **Desafío:** debe reconocer un cuerpo extraño en una radiografía.
- **Datos:** radiografías de pacientes con y sin cuerpos extraños

Machine Learning (aprendizaje automático)

#GoogleActívate

El **aprendizaje automático** o aprendizaje automatizado o aprendizaje de máquinas (del inglés, Machine Learning) es el subcampo de las ciencias de la computación y una rama de la Inteligencia Artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.

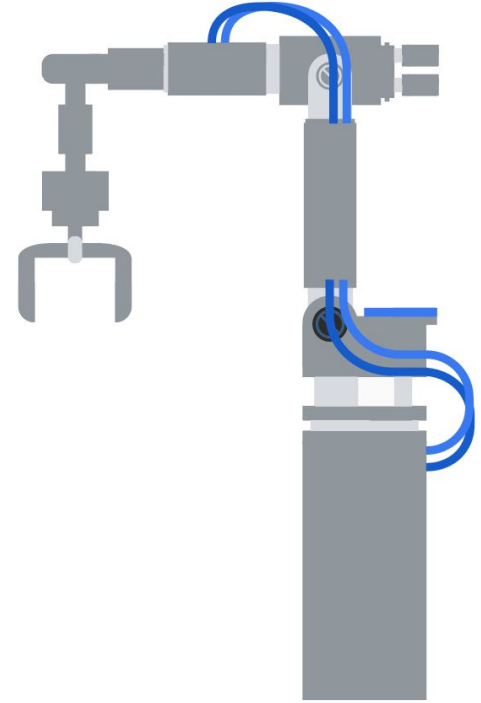


Automatización de tareas cognitivas

#GoogleActíivate

De la misma manera que en los 70 se automatizaron tareas manuales **sencillas** con robots, se están automatizando tareas cognitivas.

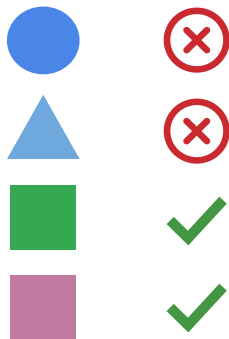
Estas tareas cognitivas son sencillas y se necesita ensamblarlas juntas para producir resultados.



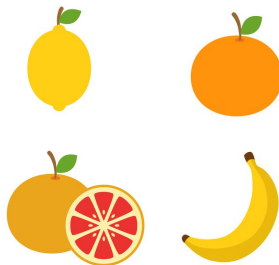
Tipos de aprendizaje

#GoogleActívale

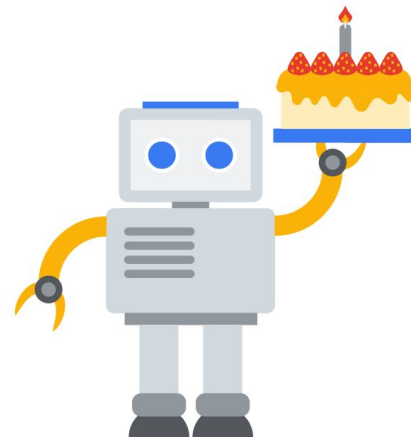
Generalización
y relación



Comparación



Refuerzo

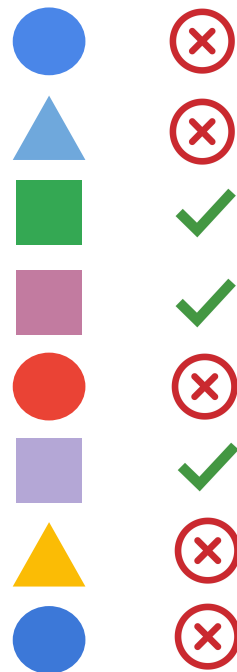


Generalización

#GoogleActívate

Una puerta solo se abre cuando un objeto adecuado es mostrado.

¿Qué tipo de objeto abre la puerta?



Generalización

#GoogleActivate

Lo primero que se hace es extraer **características** de los objetos: forma, color, tamaño, etc.



Después, buscamos una **relación** entre esas características y el **objetivo** (aceptado o no).



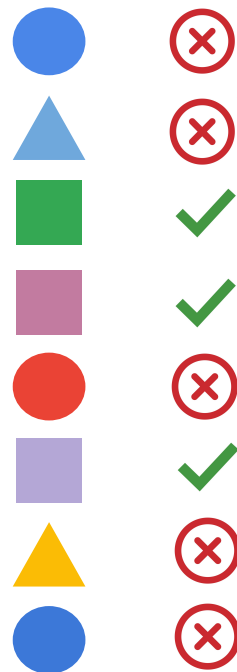
Y ya podemos predecir el **objetivo** para un objeto

Generalización

#GoogleActívate

Una puerta solo se abre cuando un objeto adecuado es mostrado.

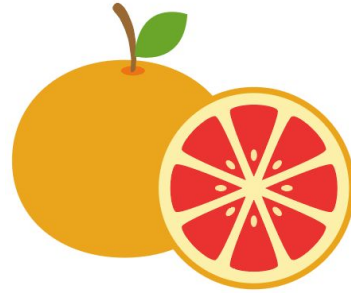
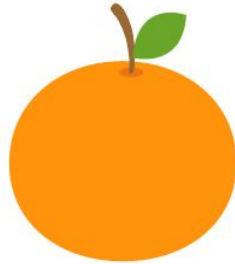
¿Qué tipo de objeto abre la puerta?



Comparación

#GoogleActívale

¿Cómo agruparíais estos objetos ?



Comparación

#GoogleActíivate

Lo primero que se hace es extraer **características** de los objetos: forma, color, tamaño, cítricos, azucarado, etc.

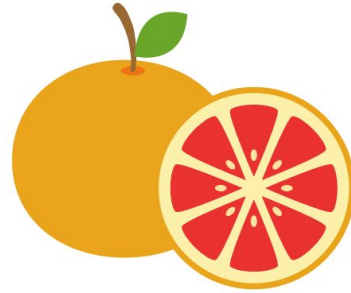
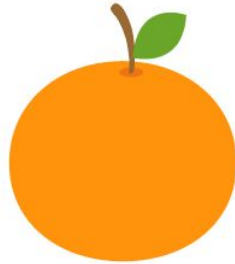


Después buscamos una **definición de similitud** entre elementos basada en esas características. **No hay objetivo.**

Comparación

#GoogleActívale

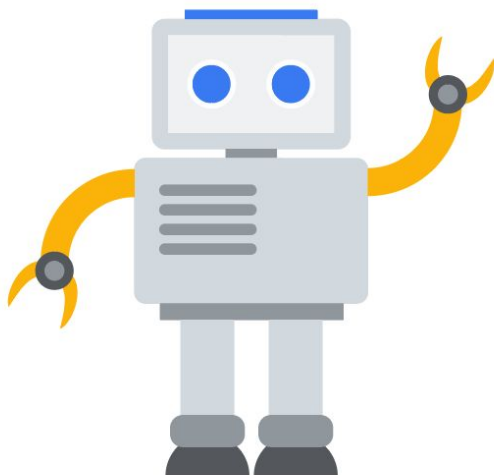
¿Cómo agruparías estos objetos ?



Recompensa

#GoogleActívate

AGENTE



Estado

Acción

Nuevo estado

Recompensa



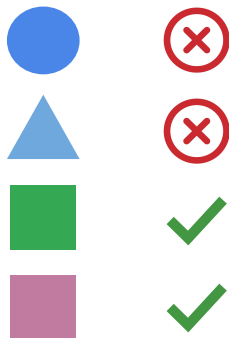
ENTORNO



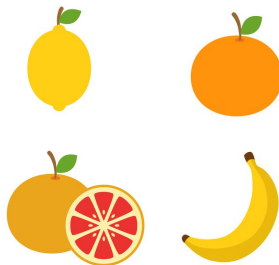
Tipos de aprendizaje

#GoogleActívate

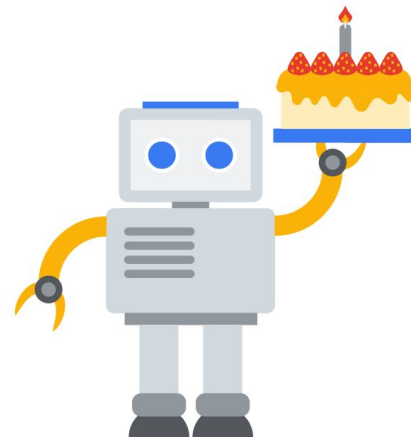
Generalización
y relación



Comparación



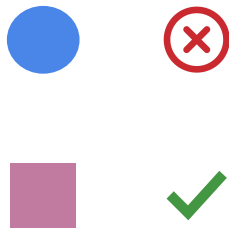
Recompensa





Aprendizaje supervisado

#GoogleActívale

A la generalización y relación se le llama **aprendizaje supervisado**.

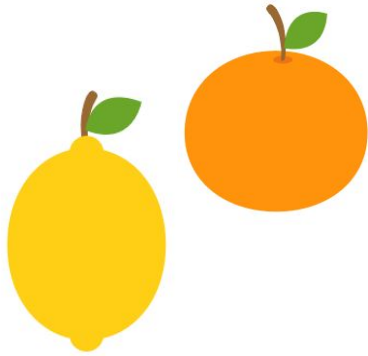


Se le llama así porque a cada elemento se le asigna una variable **objetivo** (en este caso  y .

Cuando se asigna una **etiqueta** es una **clasificación**. Cuando se asigna un **número**, es un **regresión**.

Aprendizaje no supervisado

#GoogleActívate



A la comparación se le llama **aprendizaje no supervisado**.

Se le llama así por que **NO** se le asigna ninguna variable objetivo.

Se realiza sobre todo para :

- **Segmentación** (agrupar).
- **Simplificación** de las características.

Aprendizaje por refuerzo

#GoogleActivate



Al aprendizaje por recompensa, se le denomina **aprendizaje por refuerzo**.

Principales tipos de tareas cognitivas

#GoogleActíivate

Aprendizaje supervisado (generalización de experiencia pasada)

Regresión: el objetivo es un número.



Clasificación: el objetivo es una etiqueta.

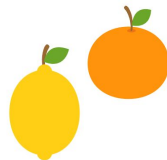


Aprendizaje no supervisado (comparando elementos, sin objetivo)

Segmentación.

Simplificación de características.

Recomendación.

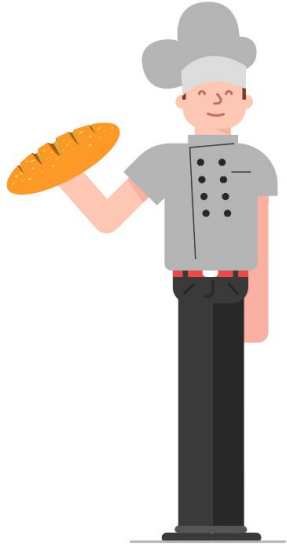


Aprendizaje reforzado (basado en recompensa)



Tareas cognitivas para cada problema

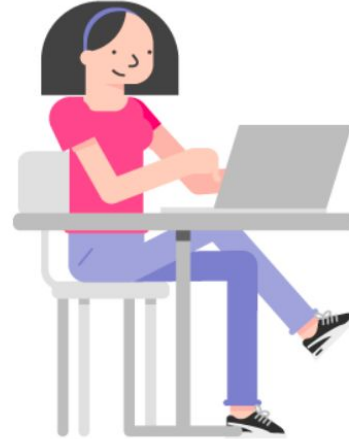
#GoogleActívate



Panadero/a:
regresión



Operario/a de fábrica:
regresión



Responsable de Marketing:
segmentación y
clasificación



Médico:
clasificación

Regression

Regresión: precio de las casas

#GoogleActivate

Problema

Queremos predecir el precio de una casa en función de sus características.

Fuentes de datos

[Catastro](#)

[Kaggle](#)

[Idealista](#)

[Fotocasa](#)

Premisa básica

#GoogleActivate

Tenemos una lista de casas con su precio. Las casas se diferencian por sus características y cada una tiene un precio.

La idea es cómo podemos "aprender" a poner un precio a las casas en función de los precios que ya se han puesto a otras casas.

Es decir, esto es equivalente a encontrar una **relación** entre características de una casa y el precio de esa casa.

Extracción de características

#GoogleActívale

El primer paso para automatizar la tarea es realizar la **extracción de características**:

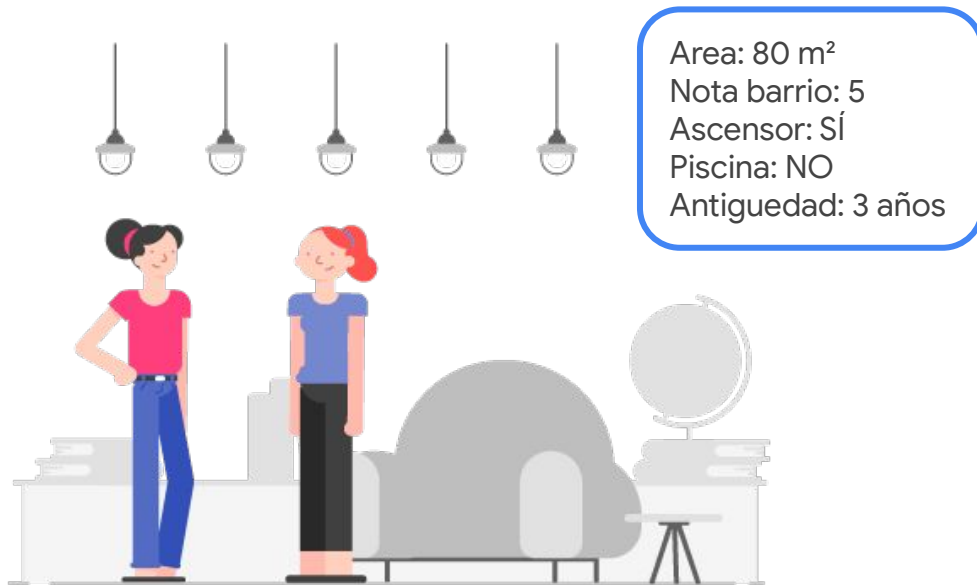


Tabla de características

#GoogleActivate

Características					Objetivo
Área	Nota barrio	Ascensor	Piscina	Antigüedad	Precio
80	5	0	1	3	150 000
50	3	1	0	15	100 000
120	3	1	0	30	200 000

Tabla de características (simple)

#GoogleActivate

Características		Objetivo
Área		Precio
80		150 000
50		100 000
120		200 000

Preparing the data

```
# Input
```

```
X = df[['TotalSF']] # pandas DataFrame
```

```
# Label
```

```
y = df["SalePrice"] # pandas Series
```

Linear Regression with Sklearn

```
# Load the library
from sklearn.linear_model import LinearRegression

# Create an instance of the model
reg = LinearRegression()

# Fit the regressor
reg.fit(X,y)

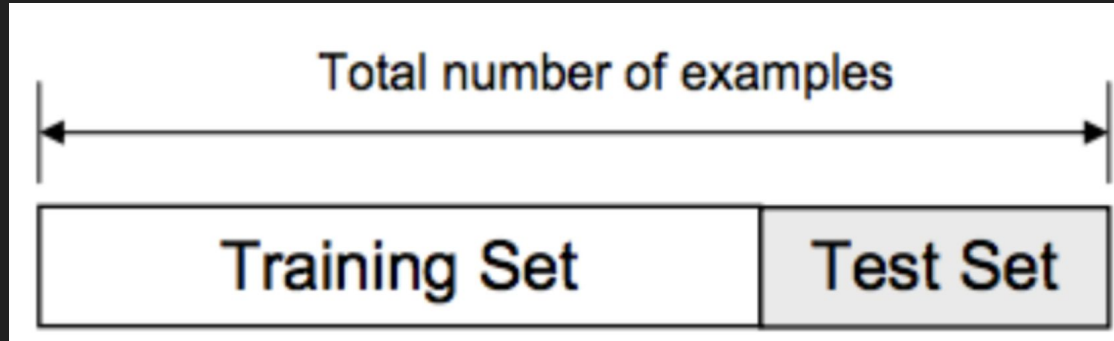
# Do predictions
reg.predict([[2540],[3500],[4000]])
```



How good is my regressor ?

In order to evaluate the regressor we just created, we would need to compare the predictions with real actual values. We are **TESTING** the regressor.

We divide our labeled original data into 2 sets: **Training** and **Testing Sets**



Train-Test Split in Sklearn

```
# Load the library
```

```
from sklearn.model_selection import train_test_split
```

```
# Create 2 groups each with input and labels
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
```

```
# Fit only with training data
```

```
reg.fit(X_train,y_train)
```


Metrics: MAE and MAPE

MAE is the sum of the absolute values of the error:

$$\frac{1}{n} \sum_{i=1}^n |y_i - x_i| = \frac{1}{n} \sum_{i=1}^n |e_i|$$

MAPE is almost the same but gives the percentage of the absolute value of error

$$\frac{1}{n} \sum_{i=1}^n \frac{|y_i - x_i|}{|y_i|} = \frac{1}{n} \sum_{i=1}^n \frac{|e_i|}{|y_i|}$$

MAE in sklearn

```
# Load the scorer

from sklearn.metrics import mean_absolute_error

# Use against predictions

mean_absolute_error(reg.predict(X_test), y_test)
```

MAPE is not in Sklearn, so we implement ourselves

```
np.mean(np.abs(reg.predict(X_test)-y_test)/y_test)
```

k Nearest Neighbors

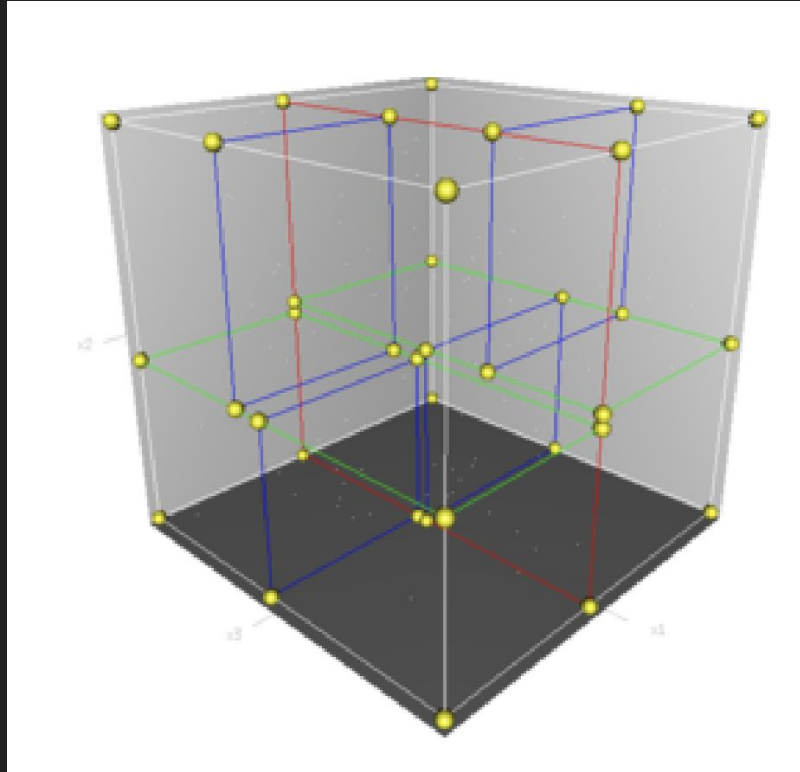
k Nearest Neighbors predicts by taking the k nearest neighbors to the input from the training data, and then combines the labels of each.

It requires that the dataset has a **DISTANCE**.

No Training Phase :) BUT it keeps all the data

Warning: if it is found that two neighbors, neighbor k+1 and k, have identical distances but different labels, the results will depend on the ordering of the training data.

k Nearest Neighbors: Data Partition



k Nearest Neighbors: Parameters

k: Number of neighbors

weight: Way to combine the label of the nearest point

Uniform: All the same

Distance: Weighted Average per distance

Custom: Weighted Average provided by user

partition: Way to partition the training dataset (ball_tree, kd_tree, brute)

k Nearest Neighbors in Sklearn

```
# Load the library

from sklearn.neighbors import KNeighborsRegressor

# Create an instance

regk = KNeighborsRegressor(n_neighbors=2)

# Fit the data

regk.fit(X,y)
```

Metric: RMSE

RMSE **penalizes more high values** of error

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - x_i)^2}{n}}$$

RMSE in Sklearn

```
# Load the scorer
```

```
from sklearn.metrics import mean_squared_error
```

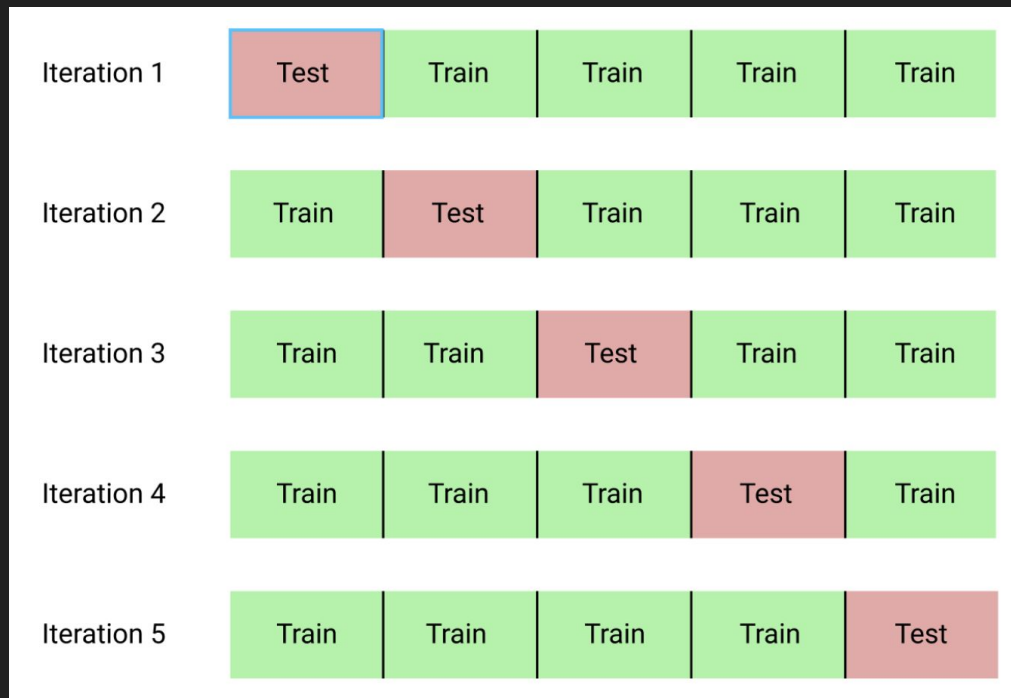
```
# Use against predictions (we must calculate the square root of the MSE)
```

```
np.sqrt(mean_squared_error(reg.predict(X_test), y_test))
```

Cross Validation

The dataset is split into n random parts. Then we iterate by:

- Training with $n-1$ chunks
- Test with the remainder
- We then can calculate mean or variance of the error.



Cross Validation in Sklearn

```
# Load the library

from sklearn.model_selection import cross_val_score

# We calculate the metric for several subsets (determine by cv)

# With cv=5, we will have 5 results from 5 training/test

cross_val_score(reg,X,y,cv=5,scoring="neg_mean_squared_error")
```

Testing Parameters: GridSearchCV

We could then try to find the best parameters by testing all of the combinations of them. We test a GRID of parameters.

```
from sklearn.model_selection import GridSearchCV

from sklearn.neighbors import KNeighborsRegressor

reg_test = GridSearchCV(KNeighborsRegressor(),

                        param_grid={"n_neighbors":np.arange(3, 50)})

# Fit will test all of the combinations

reg_test.fit(X,y)
```

Testing Parameters: GridSearchCV

```
# Fit will test all of the combinations
```

```
reg_test.fit(X,y)
```

```
# Best estimator and best parameters
```

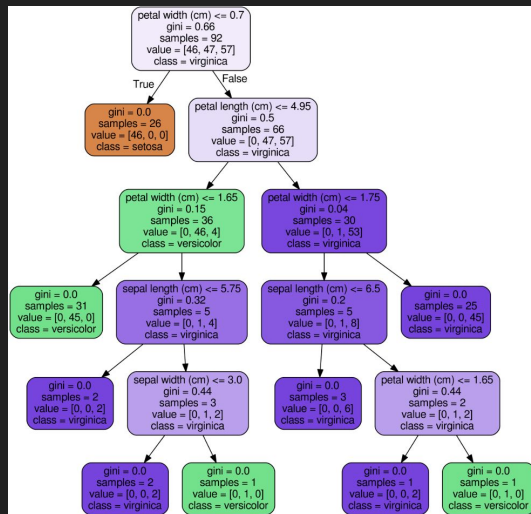
```
reg_test.best_score_
```

```
reg_test.best_estimator_
```

```
reg_test.best_params_
```

Decision Tree

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.



Decision Tree: Building homogeneous partitions

- Start at the training dataset
- For each feature:
 - Split in 2 partitions
 - Calculate the purity/homogeneity gain
- Keep the feature split with the best gain
- Repeat for the 2 new partitions

Homogeneity gain is calculated with the variance (regression) or entropy (classification).

Decision Tree: Main Parameters

Max_depth: Number of Splits

Min_samples_leaf: Minimum number of observations per leaf

Decision Tree in Sklearn

```
# Load the library

from sklearn.tree import DecisionTreeRegressor

# Create an instance

regd = DecisionTreeRegressor(max_depth=3)

# Fit the data

regd.fit(X,y)
```

Metric: Correlation

Correlation measures the correlation between the predictions and the real value.

Direct Calculation

```
np.corrcoef(reg.predict(X_test),y_test)[0][1]
```

Custom Scorer

```
from sklearn.metrics import make_scorer
```

```
def corr(pred,y_test):
```

```
    return np.corrcoef(pred,y_test)[0][1]
```

Put the scorer in cross_val_score

```
cross_val_score(reg,X,y,cv=5,scoring=make_scorer(corr))
```

Metric: Bias

Bias is the average of errors.

```
# Direct Calculation
```

```
np.mean(reg.predict(X_test)-y_test)
```

```
# Custom Scorer
```

```
from sklearn.metrics import make_scorer
```

```
def bias(pred,y_test):
```

```
    return np.mean(pred-y_test)
```

```
# Put the scorer in cross_val_score
```

```
cross_val_score(reg,X,y,cv=5,scoring=make_scorer(bias))
```

Drawing the Decision Tree

```
from IPython.display import Image

from sklearn.tree import export_graphviz

import pydotplus

dot_data = StringIO()

export_graphviz(dtree, out_file=dot_data, filled=True, rounded=True,
                special_characters=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

Bias-variance tradeoff

We must find a compromise between two sources of error:

The **bias** is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (**underfitting**).

The **variance** is error from sensitivity to small fluctuations in the training set. High variance can cause **overfitting**: modeling the random noise in the training data, rather than the intended outputs.

Classification

Problem Statement

Determine if the car should go fast or slow according to the bumpiness and slope of the route.



Logistic Regression in sklearn

```
# Load the library

from sklearn.linear_model import LogisticRegression

# Create an instance of the classifier

clf=LogisticRegression()

# Fit the data

clf.fit(X,y)
```


Metric: Accuracy

```
# With Metrics
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, clf.predict(X_test))
```

```
# Cross Validation
```

```
cross_val_score(clf, X, y, scoring="accuracy")
```

k nearest neighbor (Same Parameters)

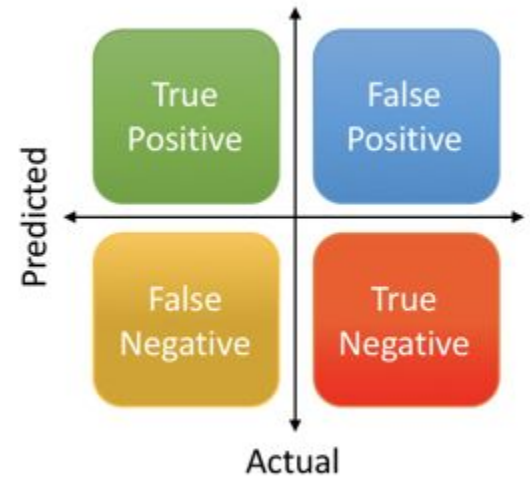
```
# Load the library
from sklearn.neighbors import KNeighborsClassifier
# Create an instance
regk = KNeighborsClassifier(n_neighbors=2)
# Fit the data
regk.fit(X,y)
```

Metric: Precision and Recall

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total}}$$



Metric: Precision and Recall

```
# Metrics

from sklearn.metrics import precision_score, recall_score

from sklearn.metrics import confusion_matrix, classification_report

precision_score(y_test,clf.predict(X_test))

classification_report(y_test,clf.predict(X_test))


# Cross Validation

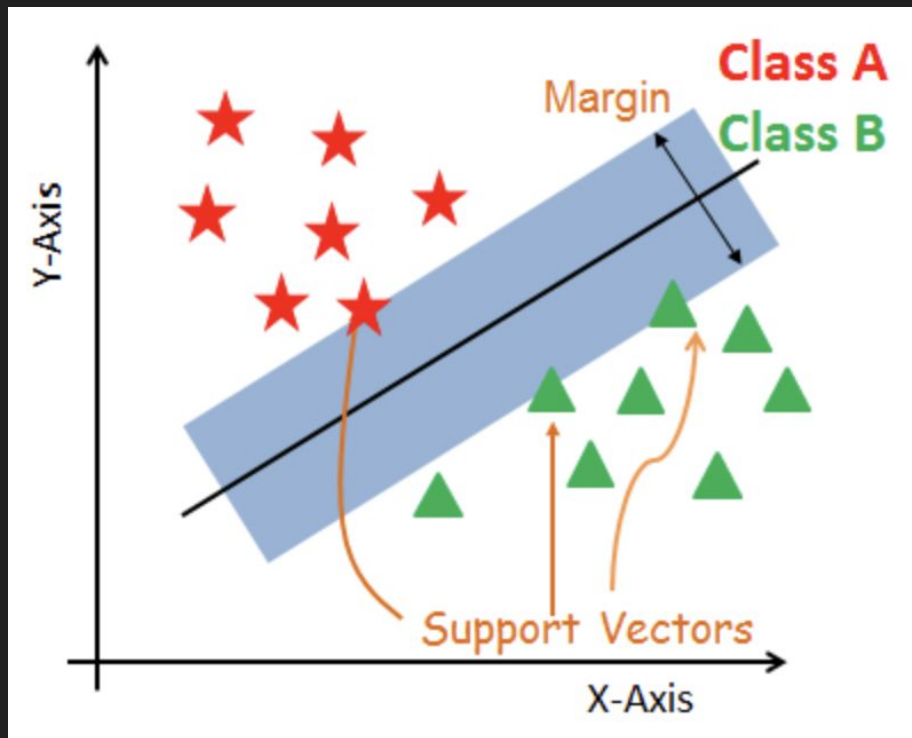
cross_val_score(clf,X,y,scoring="precision")

cross_val_score(clf,X,y,scoring="recall")
```

Support Vector Machine

Classes are separated by a line.

(See joined notebook)



Support Vector Machines: Main Parameters

C: Sum of Error Margins

kernel:

linear: line of separation

rbf: circle of separation

Additional param gamma: Inverse of the radius

poly: curved line of separation

Additional param degree: Degree of the polynome

Support Vector Machine in Sklearn

```
# Load the library
```

```
from sklearn.svm import SVC
```

```
# Create an instance of the classifier
```

```
clf = SVC(kernel="linear",C=10)
```

```
# Fit the data
```

```
clf.fit(X,y)
```

Decision Tree in Sklearn

```
# Import library

from sklearn.tree import DecisionTreeClassifier

# Create instance

clf = DecisionTreeClassifier(min_samples_leaf=20,max_depth=3)

# Fit the data

clf.fit(X,y)
```


Predict Probability

A classifier can not only predict a class. It can also predict the probability of each class.

Probability of first
instance being a 0

	0	1
0	0.350438	0.649562
1	0.916989	0.083011
2	0.224553	0.775447
3	0.921397	0.078603
4	0.166357	0.833643

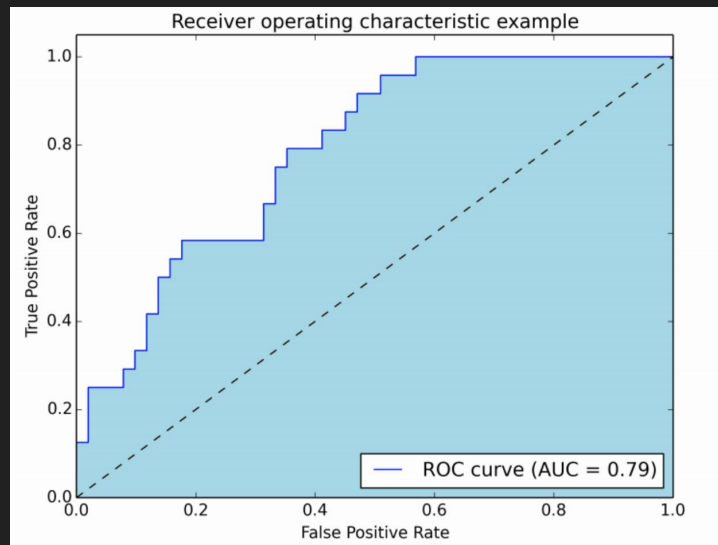
Probability of first
instance being a 1

Metric: Receiver Operating Characteristic Curve

You can change the threshold and calculate:

- Number of True Positives: Correctly predicted as 1
- Number of False Positives: Incorrectly predicted as 1

The ROC Curve shows how confident your classifier is, with the area under this curve.



ROC Curve in Python

```
# Load the library

from sklearn.metrics import roc_curve

# We chose the target

target_pos = 1 # Or 0 for the other class

fp, tp, _ = roc_curve(y_test, pred[:, target_pos])

plt.plot(fp, tp)
```

AUC metric

```
# Metrics
```

```
from sklearn.metrics import roc_curve, auc
```

```
fp, tp, _ = roc_curve(y_test, pred[:, 1])
```

```
auc(fp, tp)
```

```
# Cross Validation
```

```
cross_val_score(clf, X, y, scoring="roc_auc")
```

Saving and delivering a model

```
clf = DecisionTreeClassifier(max_depth=17)

clf.fit(X,y)

import pickle

pickle.dump(clf,open("modelo.pickle","wb"))


clf_loaded = pickle.load(open("modelo.pickle","rb"))

ndf = pd.read_csv("nuevosind.csv")

clf_loaded.predict(ndf)
```

Latex Formulas

MAE: $\frac{\sum_{i=1}^n |y_i - x_i|}{n} = \frac{\sum_{i=1}^n |e_i|}{n}$

MAPE: $\frac{\sum_{i=1}^n |y_i - x_i|}{n |y_i|} = \frac{\sum_{i=1}^n |e_i|}{n |y_i|}$

Todo

Change overfit - underfit chart

Add ways to optimize: Analytical/Gradient Descent/Genetic Algo/Grid Search