

Programación Técnica y Científica

Prof. Miguel García Silvente

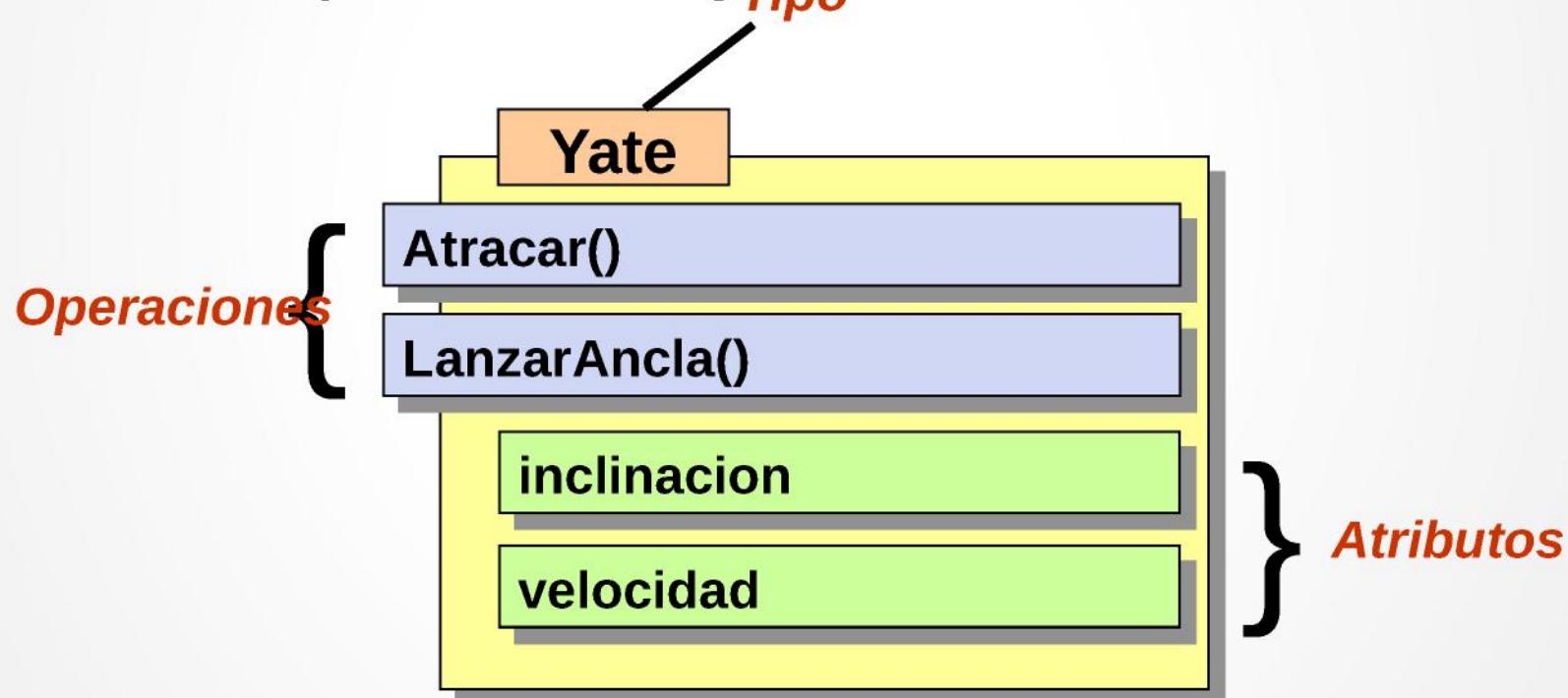
Tema 5

Python Avanzado

Programación orientada a objetos

Clases

- Una clase define un tipo de dato junto con sus operaciones y su estado.



Elementos de una clase

Una clase define un tipo de dato junto con sus operaciones y su estado.

Algunas operaciones:

`__init__` constructor

`__del__` se llama justo antes de destruir el objeto

`__repr__` la representación oficial en forma de string

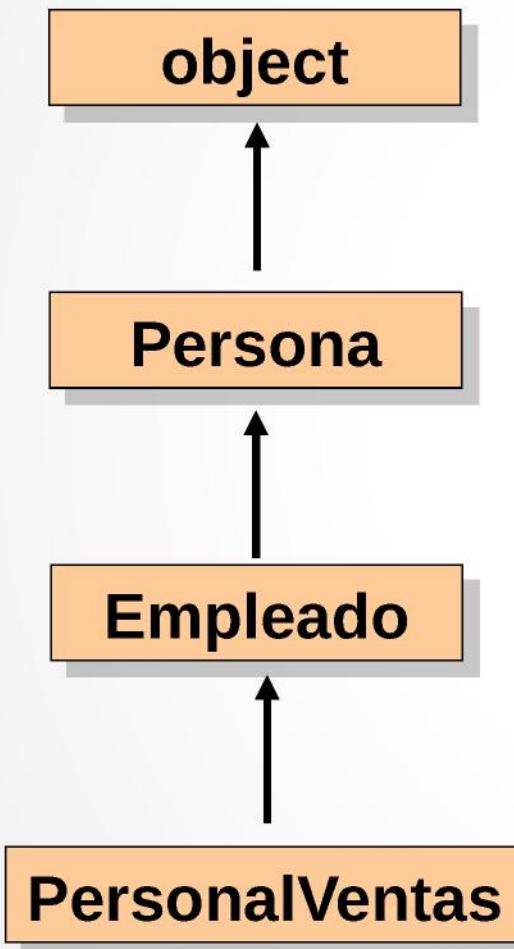
`__str__` la conversión a string

Se usa *self* para acceder al propio objeto

Ejemplo de clase

```
class Yate :  
    def __init__(self, inclinacion, velocidad) : #constructor  
        self.inclinacion = inclinacion # atributo  
        self.velocidad = velocidad # atributo  
  
    def __repr__(self): # método de la clase  
        return str(self.inclinacion)+str(self.velocidad)
```

La clase *object*



Todas las clases en Python heredan de la clase *object*

Crear objetos

- Se usa el nombre de la clase para crear un objeto nuevo
 - Devuelve una referencia al objeto
 - El objeto no es accesible de forma directa
- El objeto se destruye cuando no queda ninguna referencia.

```
class Persona :
```

```
    ...
```

```
pepe = Persona()  
juan = Persona()
```

Manipulación de objetos

```
p1 = Punto(5,7)
p2 = Punto(15,17)
p3 = Punto(25,27)

p1.desplazar(1,1)
p1.posicion()

p2.desplazar(1,10)
p2.posicion()

p3.desplazar(1,100)
p3.posicion()
# Punto.posicion(p3)
```

```
class Punto:
    cont = 0 # Atributo global a la clase
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
        Punto.cont += 1

    def posicion(self):
        return self.x, self.y

    def desplazar(self, dx, dy):
        self.x += dx
        self.y += dy

    def cuantos_puntos(self):
        return Punto.count
```

Atributos de un objeto

Los atributos se pueden definir cuando sean necesarios

Pueden aparecer sólo en un objeto

b1 tiene alto, ancho, color

b2 tiene alto, ancho

Se usa **pass** para definir una clase vacía

```
class Box:  
    pass  
b1 = Box( )  
b2 = Box( )  
b1.alto = 100  
b1.ancho = 50  
b1.color = "red"  
b2.alto = 100  
b2.ancho = 50
```

Ejemplo de herencia

```
class Punto3D(Point):
    z = 0

    def __init__(self, x, y, z):
        Punto.__init__(self, x, y)
        self.z = z

    def desplazar(self, dx, dy, dz):
        Punto.desplazar(self, dx, dy)
        self.z += dz
```

Ejemplo de clase

```
class Pila:  
    """Tipo de dato pila"""  
  
    def __init__(self):          # constructor  
        self.items = [] # una lista como representación  
  
    def push(self, x):  
        self.items.append(x)  
  
    def pop(self):  
        x = self.items[-1]  
        del self.items[-1]  
        return x  
  
    def empty(self):  
        return len(self.items) == 0
```

Ejemplo de clase

- Crear una pila

`x = Pila()`

- Uso de métodos de la clase

`x.empty()` # -> 1

`x.push(1)` # [1]

`x.empty()` # -> 0

`x.push("hello")` # [1, "hello"]

`x.pop()` # -> "hello" # [1]

- Se puede acceder a los componentes de la clase

`x.items` # -> [1]

Ejemplo de subclase

```
class PilaMejorada(Pila):
    """pila con la posibilidad de poder inspeccionar más
    valores"""

    def seleccionar(self, n):
        """seleccionar(0) devuelve top; seleccionar(-1) devuelve
        el siguiente, etc."""
        size = len(self.items)
        assert 0 <= n < size
        return self.items[size-1-n]
```

Otro ejemplo de herencia

```
class Animal:  
    def __init__(self, name):  
        self.name = name
```

```
class Gato(Animal):  
    def hablar(self):  
        return 'Miau!'
```

```
class Perro(Animal):  
    def hablar(self):  
        return 'Guau!'
```

Ejemplo de subclase

```
class PilaLimitada(PilaMejorada):
    """pila mejorada con límite en número de elementos"""

    def __init__(self, limit):
        self.limit = limit
        PilaMejorada.__init__(self) # constructor de la clase base

    def push(self, x):
        assert len(self.items) < self.limit
        PilaMejorada.push(self, x) # llamada al "super" método
```

Restricciones de acceso (I)

Todos los datos son públicos aunque se usa la convención de usar
__<dato> para indicar que es privado

```
class Ejemplo :  
    def __init__(self):  
        self.pub = 0  
        self.__priv = 1  
        self.__priv2 = 2  
    def contador(self):  
        Ejemplo.cont += 1  
        return Ejemplo.cont
```

```
ej = Ejemplo()  
print(ej.__priv, ej.__priv2)
```

AttributeError: 'Ejemplo' object has no attribute '__priv2'

Prof. Miguel García Silvente

17

Restricciones de acceso (II)

Para acceder al dato “privado” anterior:

```
class Ejemplo :  
    def __init__(self):  
        self.pub = 0  
        self.__priv = 1  
        self.__priv2 = 2  
    def contador(self):  
        Ejemplo.cont += 1  
        return Ejemplo.cont
```

```
ej = Ejemplo()  
print(ej.__priv2) # permite el acceso
```

Parámetros de distinto tipo

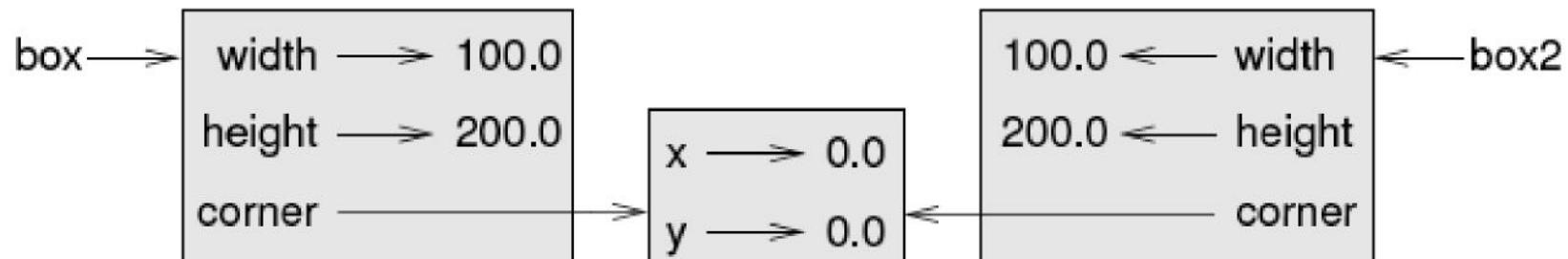
```
def increment(self, t2):
    if type(t2) == Time:
        self.hour += t2.hour
        self.minute += t2.minute
        self.second += t2.second
    elif type(t2) == int:
        # Incrementa los segundos
        self.second += t2
    else:
        raise AttributeError, \
            'invalid argument passed to Time.increment()'
    self.adjust_base_60()
```

Limitaciones de copy

`box2 = box.copy()`

Se resuelve con

`box3 = copy.deepcopy(box)`



Métodos intrínsecos (I)

__dict__ Diccionario que permite conocer los atributos de la clase y sus valores, así como modificarlos

```
for i in ej.__dict__.keys( ):  
    print(i, ej.__dict__[i])
```

Ejemplo_priv2

```
pub 0
```

```
_priv 1
```

Métodos intrínsecos (II)

`__call__` permite usar un objeto como una función

```
class Factorial:
```

```
    def __init__(self):
        self.cache = {}
    def __call__(self, n):
        if n not in self.cache:
            if n == 0:
                self.cache[n] = 1
            else:
                self.cache[n] = n * self.__call__(n-1)
        return self.cache[n]
```

```
fact = Factorial()
```

```
for i in xrange(10):
```

```
    print("{}! = {}".format(i, fact(i)))
```

Sobrecarga de operadores: datos numéricos

```
class Tiempo:  
    def __init__(self, hrs, min):  
        self.hrs = hrs  
        self.min = min  
  
    def __add__(self, t):  
        hrs = self.hrs + t.hrs  
        min = self.min + t.min  
        if min >= 60:  
            hrs = hrs + 1  
            min = min - 60  
        return Tiempo(hrs, min)  
  
    def __repr__(self):  
        return "Tiempo es " + str(self.hrs) + str(self.min))
```

t3 = t1 + t2

Sobrecarga de operadores: datos numéricos

```
class Tiempo:  
    def __init__(self, hrs, min):  
        self.hrs = hrs  
        self.min = min  
  
    def __iadd__(self, t):  
        self.hrs = self.hrs + t.hrs  
        self.min = self.min + t.min  
        if self.min >= 60:  
            self.hrs = self.hrs + 1  
            self.min = self.min - 60  
        return self  
  
    def __repr__(self):  
        return "Tiempo es" + str(self.hrs) + str(self.min))
```

t1 += t3

Sobrecarga de operadores: datos numéricos

```
class Tiempo:  
    def __init__(self, hrs, min):  
        self.hrs = hrs  
        self.min = min  
  
    def __add__(self, other):  
        hrs = self.hrs + other.hrs  
        min = self.min + other.min  
        if min >= 60:  
            hrs = hrs + 1  
            min = min - 60  
        return Tiempo(hrs, min)  
  
    def __iadd__(self, t):  
        self.hrs = self.hrs + t.hrs  
        self.min = self.min + t.min  
        if self.min >= 60:  
            self.hrs = self.hrs + 1  
            self.min = self.min - 60  
        return self  
  
    def __repr__(self):  
        return "Tiempo es" + str(self.hrs) + str(self.min))
```

```
t1 = Tiempo(5,30)  
t2 = Tiempo(3,30)  
t3 = t1 + t2  
print(t3)  
  
t1 += t3  
print(t1)
```

Tiempo es 9 0

Tiempo es 14 30

Sobrecarga de operadores

Operador	Método de la clase
-	<code>__sub__(self, other)</code> <code>__rsub__(self, other)</code>
+	<code>__add__(self, other)</code> <code>__radd__(self, other)</code>
*	<code>__mul__(self, other)</code> <code>__rmul__(self, other)</code>
/	<code>__div__(self, other)</code> <code>__rdiv__(self, other)</code>
%	<code>__mod__(self, other)</code> <code>__rmod__(self, other)</code>
<u>Operadores unarios</u>	
-	<code>__neg__(self)</code>
+	<code>__pos__(self)</code>

Operador	Método de la clase
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
>=	<code>__ge__(self, other)</code>

`rsub` es `reverse sub`, `radd` `reverse add`, ...

Prof. Miguel García Silvente

Sobrecarga de operadores: otros tipos

Se puede emular el comportamiento de otros datos

Secuencias

Troceados

Diccionarios

Métodos:

por ejemplo para diccionarios

`__len__(self)`

`__getitem__(self,key)`

`__setitem__(self,key,value)`

`__delitem__(self,key)`

Sobrecarga para tipo list

```
class MiLista:  
    def __init__(self, p1, p2):  
        self.theList = []  
        self.theList.append(p1)  
        self.theList.append(p2)  
  
    def __add__(self, other):  
        result = []  
  
        for item in self.theList:  
            result.append(item)  
  
        for item in other.theList:  
            result.append(item)  
  
        return result
```

```
t1 = MiLista(1,2)  
t2 = MiLista(3,4)  
t3 = t1 + t2
```

```
t1: [1,2]  
t2: [3,4]  
t3: [1,2,3,4]
```

Sobrecarga de [] usandogetitem

```
class misdatos:  
    def __init__(self, datos = "ABCDEFG"):  
        self.data = datos  
  
    def __getitem__(self, i):  
        return self.data[i]  
  
obj = misdatos() # también es válido obj = "ABCDEFG"  
  
theChar = obj[1]  
print (theChar)  
  
for item in obj:  
    print (item, end="")
```

B
A B C D E F G

Atributos no definidos

Llamar a un método no definido genera una excepción.

A menos que se defina `__getattr__()`

```
class MiClase :  
    def __getattr__(self, method):  
        print ("método no encontrado: ", method)  
        return self.myDefault  
  
    def myDefault(self): print ("default()")  
    def f(self): print ("f()")  
    def g(self): print ("g()")  
    def h(self): print ("h()")
```

```
a = MiClase()  
a.f()  
a.g();  
a.h();  
a.dummy()
```

Métodos estáticos

Son comunes a todos los objetos de la clase.

Se indican con `@staticmethod`

Ejemplo:

```
class A :  
    def foo(self,x):  
        print "executing foo(%s,%s)"%(self,x)  
    @classmethod  
    def class_foo(cls,x):  
        print "executing class_foo(%s,%s)"%(cls,x)  
    @staticmethod  
    def static_foo(x):  
        print "executing static_foo(%s)"%x  
a=A()
```

Clases abstractas I

No implementan algunos métodos y se definen como abstractos

```
from abc import ABCMeta, abstractmethod, abstractproperty
class Formas:
    __metaclass__ = ABCMeta

    @abstractmethod
    def display(self): pass

    @abstractproperty
    def nombre(self): pass
```

Clases abstractas II

Se deben implementar todos los métodos abstractos

```
class Circulo(Formas):
    def __init__(self, nombre):
        self.nom = nombre

    def display(self):
        print ("Círculo", self.nombre)

    @property
    def nombre(self):
        return (self.nom)
```

Clases de tipo excepción

Se pueden definir nuevas excepciones

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
```

```
try:
    raise MyError(2*2)
except MyError as e:
    print('excepción generada, valor:', e.value)
```

excepción generada, valor: 4

```
>>> raise MyError('oops!')
```

Traceback (most recent call last):

```
  File "<stdin>", line 1, in ?
```

```
__main__.MyError: 'oops!' Prof. Miguel García Silvente
```

Iteradores

- Crea una clase con los métodos `__iter__()` y `next()`
- Se llama a `__iter__()` al principio
normalmente devuelve self
- `next()` en cada iteración

```
class Fibonacci:
```

```
    def __init__(self):  
        self.x, self.y = 0, 1  
  
    def __iter__(self):  
        return self  
  
    def next(self):  
        if self.x > 10000:  
            raise StopIteration # fin de la iteración  
  
        self.x, self.y = self.y, self.x + self.y  
  
        return self.x
```

```
for f in Fibonacci():  
    print(f)
```

Iteradores con generadores

```
class Fibonacci:  
    def __init__(self):  
        self.x,self.y = 0,1  
    def __iter__(self):  
        return self  
    def miGenerador(self):  
        while self.x < 10000:  
            yield self.x  
            self.x, self.y = self.y, self.x + self.y  
        return
```

```
f = Fibonacci()  
  
# get generator  
g = f.miGenerador()  
  
print (g.next())
```

Otros aspectos de Python

Parámetros en línea de órdenes

Los parámetros con accesibles usando una lista argv

```
import sys
print ("nombre del script: ", sys.argv[0])
print ("número de argumentos: ", len(sys.argv))
print ("argumentos: " , str(sys.argv))
```

Existe un módulo para gestionarlos: optparse

```
from optparse import OptionParser
[...]
parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  # store_false ya existe y asigna false a verbose si aparece
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")
(options, args) = parser.parse_args()
>>> ejemplo_optparse.py -f kk.txt hola
{'verbose': True, 'filename': 'kk.txt'} ['hola']
```

Análisis de redes y grafos

- Biblioteca NetworkX (<https://networkx.github.io/>) <http://networkx.lanl.gov/index.html>
- Ejemplo:

```
import networkx as nx  
G = nx.Graph()  
G.add_node("spam")  
G.add_edge(1,2)  
print(G.nodes())  
[1, 2, 'spam']  
print(G.edges())  
[(1, 2)]  
nx.draw(G)  
nx.draw_networkx(G, with_labels=True)  
matplotlib.pyplot.show()
```

Tipos de grafos

- Graph: no dirigido
- DiGraph: dirigido
- MultiGraph: no dirigido con aristas paralelas
- MultiDiGraph: dirigido con aristas paralelas
- Conversiones con `to_undirected()`, `to_directed()`
- Construcción con: `g = nx.Graph()`
- Para grafos dispersos existe el módulo `scipy.sparse.csgraph`
- Ejemplo: juego word ladder

Multihebra (I)

```
import thread
import time

def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time()) )

try:
    thread.start_new_thread( print_time, ("Thread-1", 2, ) )
    thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: unable to start thread"
```

while 1:
 pass

Prof. Miguel García Silvente

41

http://www.tutorialspoint.com/python/python_multithreading.htm

Multihebra (II)

```
import threading
import time
import logging

def non_daemon():
    time.sleep(5)
    print 'Test non-daemon'

t = threading.Thread(name='non-daemon', target=non_daemon)

t.start()

print ('Test one')
#t.join()
print ('Test two')
t.join()
```

Bases de datos

```
import MySQLdb

# conexión a base de datos
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# se prepara un cursor con cursor()
cursor = db.cursor()
# ejecuta una consulta SQL
cursor.execute("SELECT VERSION()")
# Obtiene una fila
data = cursor.fetchone()

print ("Version : %s " % data)

db.close()
```

Insertar datos en base de datos

```
import MySQLdb
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
cursor = db.cursor()

sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
      LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
    cursor.execute(sql)
    db.commit()
except:
    db.rollback()

db.close()
```

Aprendizaje automático y minería de datos

Aprendizaje automático y minería de datos

Paquetes y herramientas:

- Scikit-learn
- Orange
- Pandas
- MLpy
- MDP
- PyBrain

Scikit-learn

- Python Machine learning
- Se usa `import sklearn`
- Aprendizaje supervisado. Ejemplos etiquetados.
 - Clasificación (binaria o multiclase).
 - Regresión.
- Aprendizaje no supervisado.
 - Clustering

Conjunto de datos de ejemplo

- Dentro del paquete sklearn

```
from sklearn import datasets  
iris = datasets.load_iris()  
digits = datasets.load_digits()
```

- Un dataset internamente es un diccionario que tiene datos y metadatos
- Datasets externos

<http://scikit-learn.org/stable/datasets/index.html#external-datasets>

Data del dataset

Un array de n_samples x n_features

```
print(digits.data)
```

```
[[ 0.  0.  5. ...,  0.  0.  0.]
```

```
[ 0.  0.  0. ..., 10.  0.  0.]
```

```
[ 0.  0.  0. ..., 16.  9.  0.]
```

```
...,
```

```
[ 0.  0.  1. ...,  6.  0.  0.]
```

```
[ 0.  0.  2. ..., 12.  0.  0.]
```

```
[ 0.  0. 10. ..., 12.  1.  0.]]
```

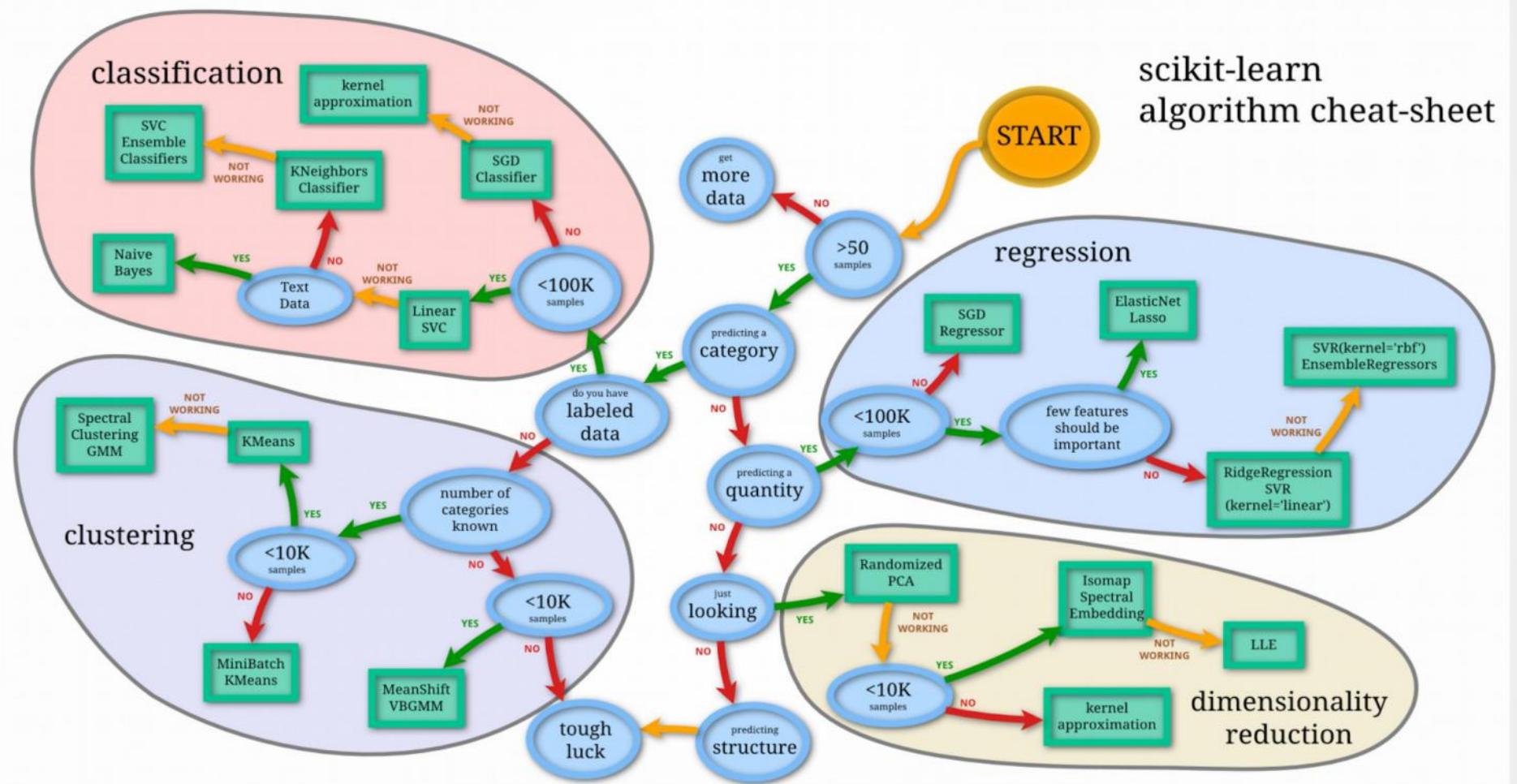
<http://scikit-learn.org/stable/tutorial/basic/tutorial.html>

Target del dataset

```
print(digits.target)
array([0, 1, 2, ..., 8, 9, 8])

>>> digits.images[0]
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

Algoritmos de scikit-learn



Aprendizaje y predicción

- Un estimador para clasificación es un objeto con los métodos:
 - `fit(X, y)`
 - `predict(T)`
- Por ejemplo para SVM
 - `sklearn.svm.SVC`
 - Ejemplo:

```
from sklearn import svm  
clf = svm.SVC(gamma=0.001, C=100.)
```

Ejemplo: Iris Dataset

Repositorio UC Irvine (UCI) Machine Learning

Clasifica los tipos de flores usando los siguientes rasgos:

- Longitud del sépalo.
- Ancho del sépalo.
- Longitud del pétalo.
- Ancho del pétalo.



Iris Setosa



Iris Virginica



Iris Versicolor

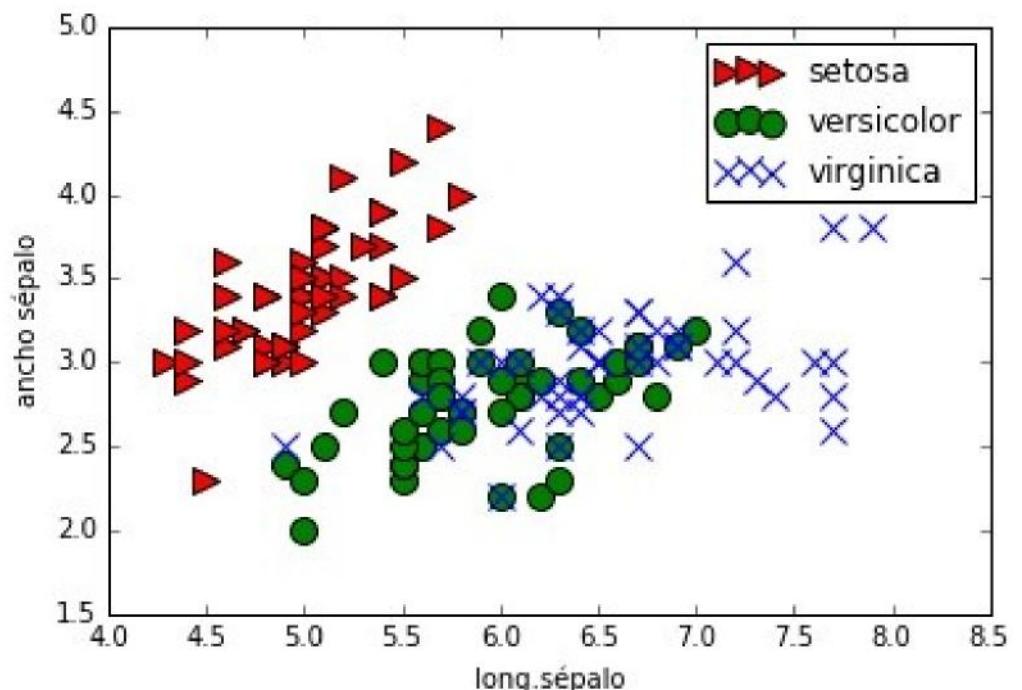
Visualización de datos

```
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
import numpy as np

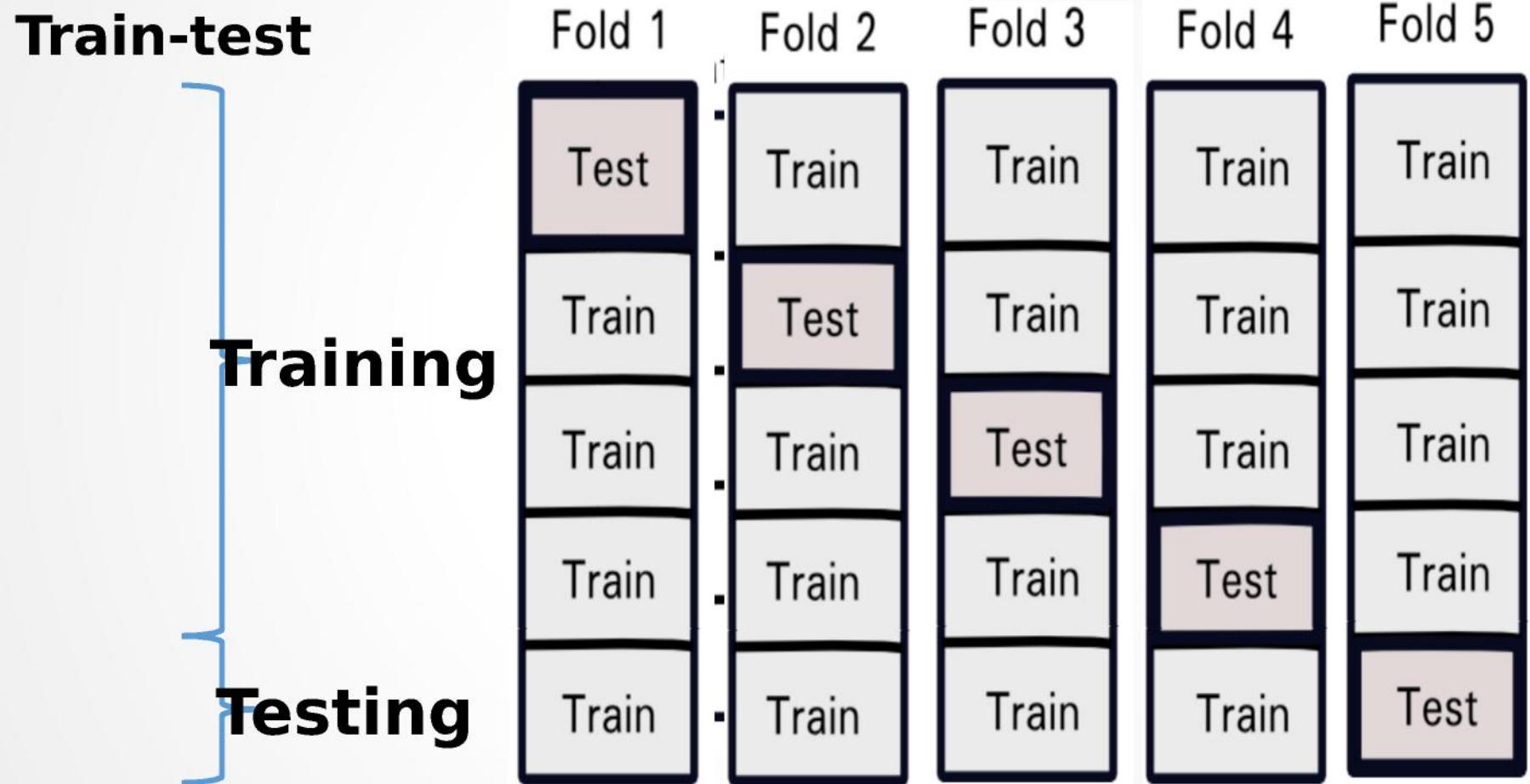
data = load_iris()
features = data['data']
feature_names = data['feature_names']
target = data['target']

cad = []
cad2 = []

for t,marker, c in zip(range(3), ">ox", "rgb") :
    aux = plt.scatter(features[target == t, 0],
                      features[target == t, 1],
                      marker = marker,
                      c=c,
                      s=100)
    cad.append(aux)
    cad2.append(data.target_names[t])
plt.legend(cad, cad2, ncol=1, loc='upper right', )
plt.show()
```



Validación cruzada (I)

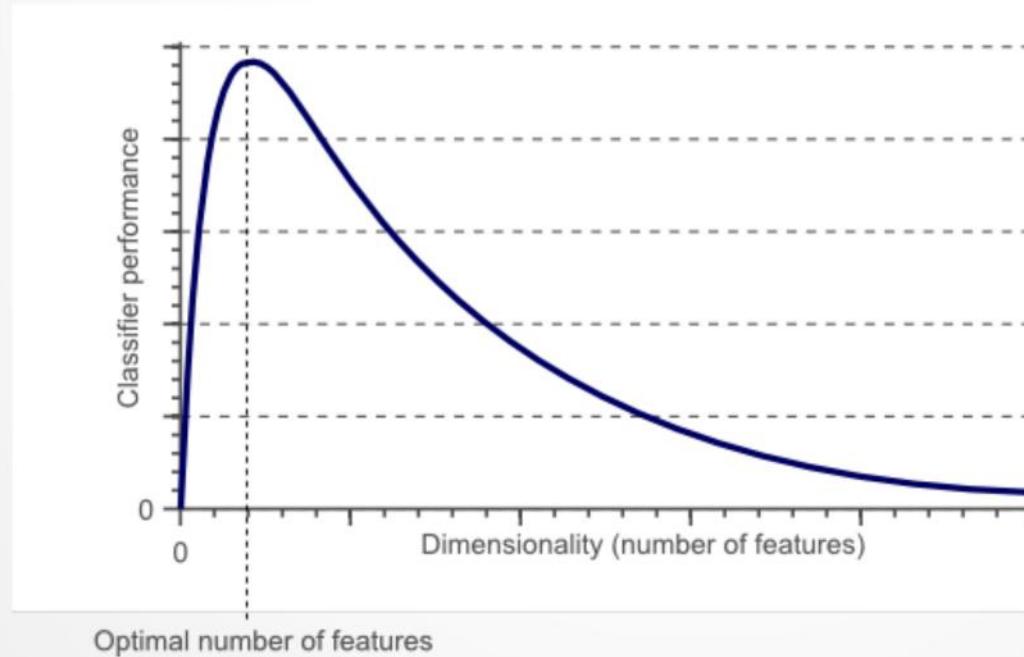


Validación cruzada (II)

- La exactitud del ajuste no debe “depender de la suerte”
`kf = KFold(n=len(binary_target), n_folds=5, shuffle=True)`
- En kf tendríamos las n_folds divisiones
`for tr, tst in kf:`
 `tr_features = features[tr, :]`
 `tr_target = binary_target[tr]`
 `tst_features = features[tst, :]`
 `tst_target = binary_target[tst]`
- Nunca se deben usar datos de entrenamiento para test

La maldición de la dimensionalidad

- Si el número de variables no es suficiente no es posible conseguir una calidad deseable.
- Si el número es demasiado alto puede ocurrir lo mismo



Número de rasgos

- Dependerá del problema y del tipo de clasificador.
- Si tienden a sobreajustar se deben usar relativamente pocos rasgos. Ej: redes neuronales, kNN, árboles de decisión.
- Por el contrario, si el método generaliza bien, se deben usar muchos rasgos. El: naive Bayes, clasificadores lineales.
- ¿Cómo seleccionar el número óptimo?
https://en.wikipedia.org/wiki/Feature_selection

Reducción de dimensionalidad

- Selección de rasgos.
- Extracción de rasgos.
 - PCA (Análisis de componentes principales)
 - Kernel PCA
 - Kernel PCA basada en grafos
 - LDA (Análisis discriminante lineal)
 - GDA (Análisis discriminante generalizado)

PCA

- Se puede definir un número de componentes principales con las que quedarnos.

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=n_components,  
          svd_solver='randomized', whiten=True).  
pca = pca.fit(X_train)
```

- Se aplica a los datos antes de usar el clasificador

```
X_train_pca = pca.transform(X_train)  
X_test_pca = pca.transform(X_test)
```

Algoritmo

1. Reescalar, estandarizar, normalizar o binarizar las entradas.
2. Si el número de rasgos es muy alto, reducir la dimensionalidad.
3. Estimación de parámetros y Validación cruzada.
4. Realizar el ajuste.
5. Medir la calidad del ajuste

Reescalar (entre 0 y 1)

```
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8] # separate array into input and output components
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Normalizar (media 0 y stddev 1)

```
from sklearn.preprocessing import StandardScaler
import pandas
import numpy
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Regresión

- Normalizar datos

```
features -= np.mean(features, axis=0)  
features /= np.std(features, axis=0)
```

- División en entrenamiento (tr) y test (tst)

```
tr_features = features[tr, :]  
tr_target = binary_target[tr]  
tst_features = features[tst, :]  
tst_target = binary_target[tst]
```

#Etiquetado binario
is_versicolor = target == 1
binary_target = np.zeros(len(target))
binary_target[is_versicolor] = 1

- Ajuste

```
model = LogisticRegression()  
model.fit(tr_features, tr_target)
```

- Exactitud del ajuste

```
tr_accuracy = np.mean(model.predict(tr_features) == tr_target)  
tst_accuracy = np.mean(model.predict(tst_features) == tst_target)
```

Clasificador SVM

- Para usar clasificación con SVM

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

...

```
model = SVC()
```

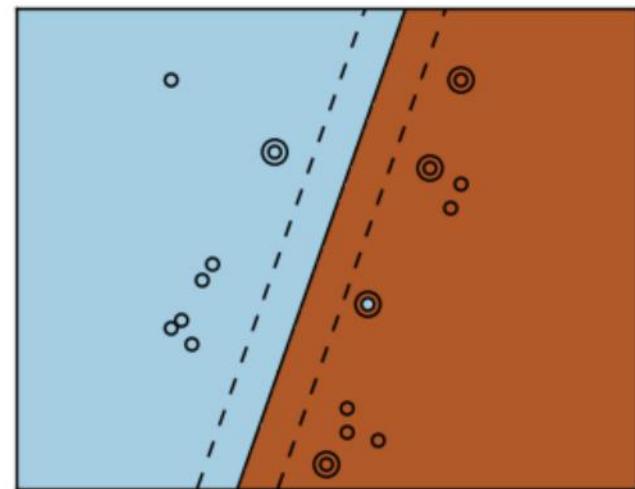
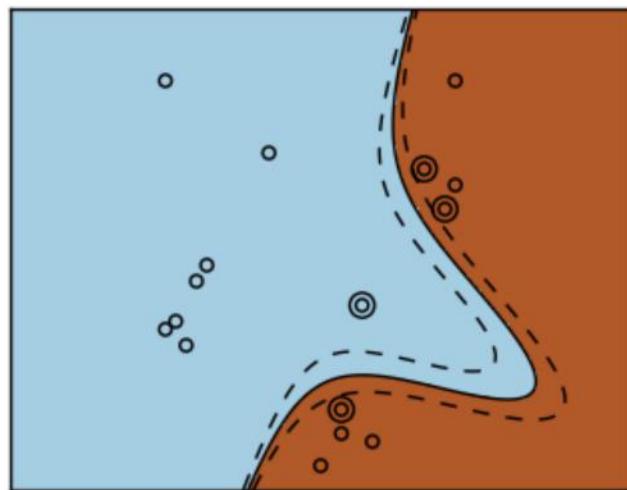
```
model.fit(tr_features, tr_target)
```

- Tiene varios parámetros (qué valores usamos?):

- **C**: parámetro de penalización para puntos “no separables”.
- **Kernel**: rbf, poly, linear.
- **Degree**: para el kernel poly.
- **Gamma**: para el kernel rbf.
- ...

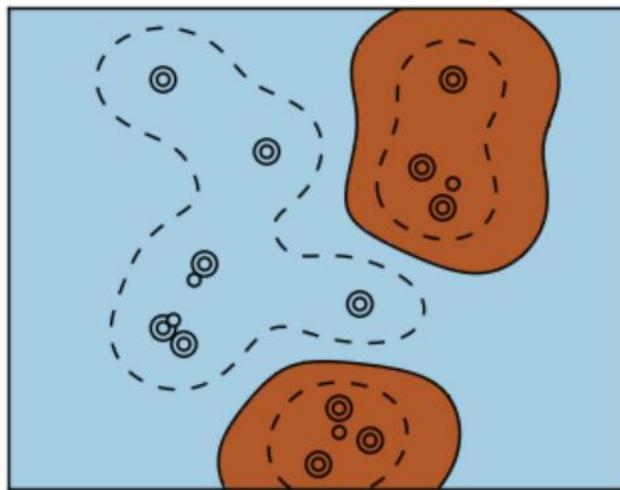
Kernel de SVM (I)

```
svc = svm.SVC(kernel='linear')
```



```
svc = svm.SVC(kernel='poly', degree=3)  
# degree: grado del polinomio
```

Kernel de SVM (II)



RBF kernel (Radial Basis Function)

`svc = svm.SVC(kernel='rbf')`

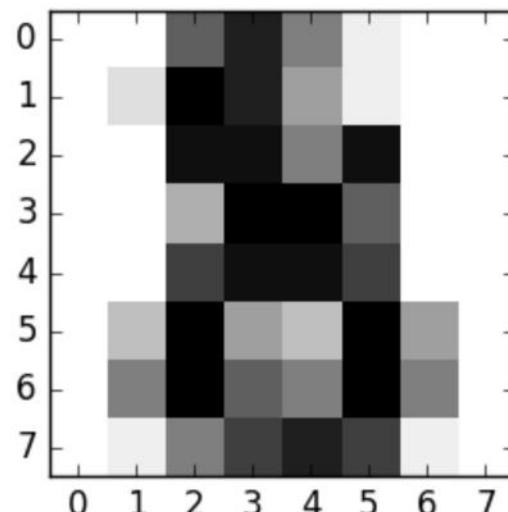
gamma: inversa del tamaño del radio del kernel

Ajuste de los datos

```
>>> clf.fit(digits.data[:-1], digits.target[:-1])  
SVC(C=100.0, cache_size=200,  
class_weight=None, coef0=0.0,  
decision_function_shape=None, degree=3,  
gamma=0.001, kernel='rbf', max_iter=-1,  
probability=False, random_state=None,  
shrinking=True, tol=0.001, verbose=False)
```

Ejemplo: clasificar dígitos

```
>>> clf.predict(digits.data[-1:])  
array([8])
```



http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py

Ejemplo: clasificar datos

```
import numpy as np  
  
data = np.array([(23, 12) ,(45, 3) ,(67, 4)  
(32,12) ,(76, 43) ,(12, 3)])  
  
target = np.array([0,0,1,1,0,0,])  
  
clf = svm.SVC().fit(data, target)  
  
X= np.array([(23,12)])  
  
print(clf.predict(X))
```

Parámetros del modelo (I)

- En el ejemplo, gamma aparece con un valor fijo. Pero puede haber mejores valores
- Los algoritmos tienen parámetros que se pueden estimar:
 - Grid search
 - Random search
- Ejemplo:

```
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],  
             'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }  
  
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),  
                   param_grid)  
  
clf = clf.fit(X_train_pca, y_train)  
  
print("Best estimator found by grid search:")  
  
print(clf.best_estimator_)
```

Parámetros del modelo (II)

```
import numpy as npfrom scipy.stats import uniform as sp_rand
from sklearn import datasets
from sklearn.linear_model import Ridge
from sklearn.grid_search import RandomizedSearchCV
# load the diabetes datasets
dataset = datasets.load_diabetes()
# prepare a uniform distribution to sample for the alpha parameter
param_grid = {'alpha': sp_rand()}
# create and fit a ridge regression model, testing random alpha values
model = Ridge()
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100)
rsearch.fit(dataset.data, dataset.target)
print(rsearch)
# summarize the results of the random parameter search
print(rsearch.best_score)
print(rsearch.best_estimator_.alpha)
```

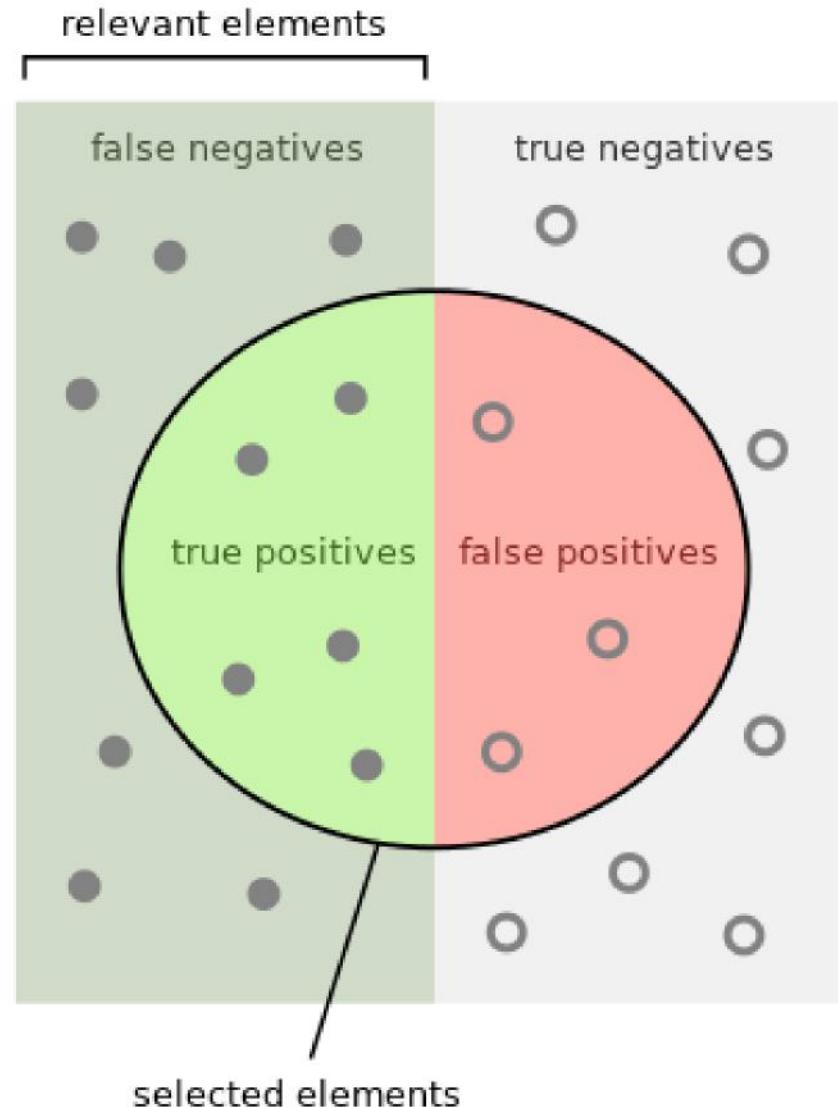
Guardar y recuperar un ajuste

```
import pickle  
  
s = pickle.dumps(clf)  
  
clf2 = pickle.loads(s)  
  
print(clf2.predict(X[0:1]))  
  
array([0])  
  
print(y[0])  
  
0
```

- Es más eficiente usar **joblib** con grandes cantidades de datos

```
from sklearn.externals import joblib  
  
joblib.dump(clf, 'filename.pkl')  
  
clf = joblib.load('filename.pkl')
```

Calidad de la clasificación



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{selected elements}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{relevant elements}}$$

Medidas de calidad (I)

F1 score: Es la media harmónica entre precisión y sensibilidad (recall)

$$F1 = \frac{2TP}{2TP + FP + FN}$$

```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
print(accuracy_score(y_true, y_pred))
print(accuracy_score(y_true, y_pred, normalize=False))
```

http://scikit-learn.org/stable/modules/model_evaluation.html#model-evaluation

Medidas (II)

```
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf = svm.SVC(probability=True, random_state=0)
print(cross_val_score(clf, X, y, scoring='accuracy'))
```

```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lasso = linear_model.Lasso()
print(cross_val_score(lasso, X, y))
[ 0.33150734  0.08022311  0.03531764]
```

Medidas (III)

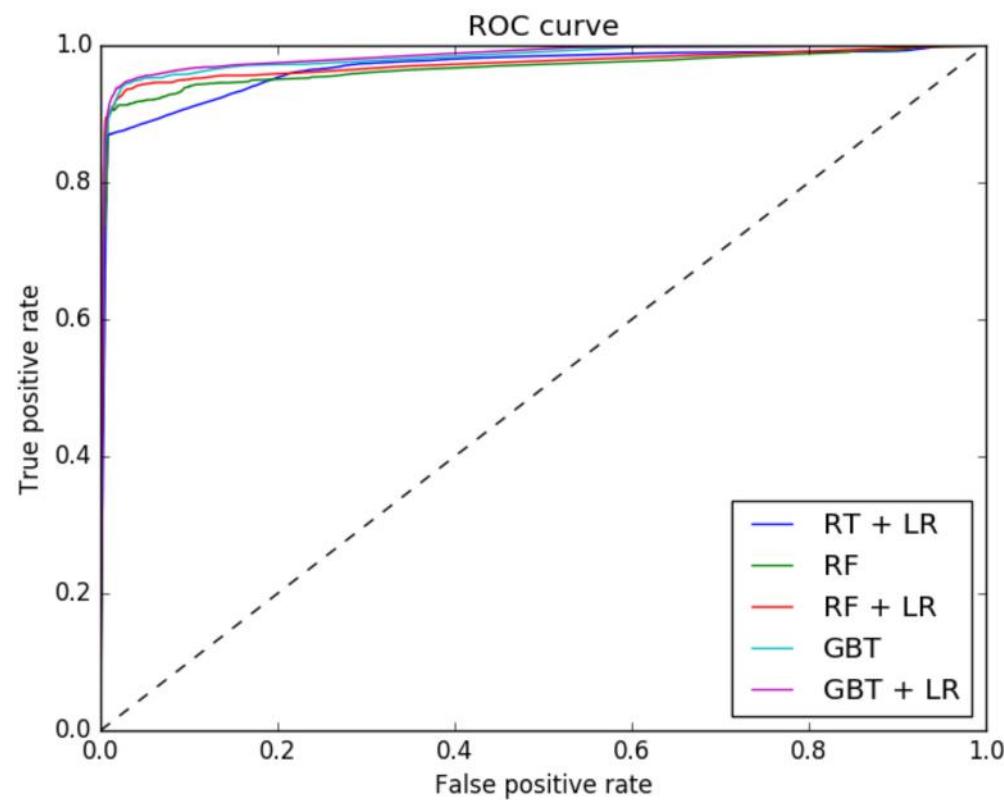
```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
print(accuracy_score(y_true, y_pred))
print( accuracy_score(y_true, y_pred,
normalize=False))
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_true, y_pred))
```

<http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>

Medidas (IV)

ROC: Receiver operating characteristic
“Detección frente a falsa alarma”



Datos de formato csv

```
import pandas as pd  
from sklearn import svm  
data = pd.read_csv(open('prueba.csv'))  
print (type(data))  
target = data["clase"]  
del data["clase"]  
clf = svm.SVC().fit(data, target)  
print(clf.predict([1,5,21]))  
print(clf.predict([10,4,6]))
```

Resources

- **LIBSVM and LIBLINEAR**

- Chih-Jen Lin, National Taiwan University.
- Simple and easy-to-use support vector machines tool.
- Hsu, C.W., Chang, C.C. and Lin, C.J., 2003.

A practical guide to support vector.
classification.

Mlight

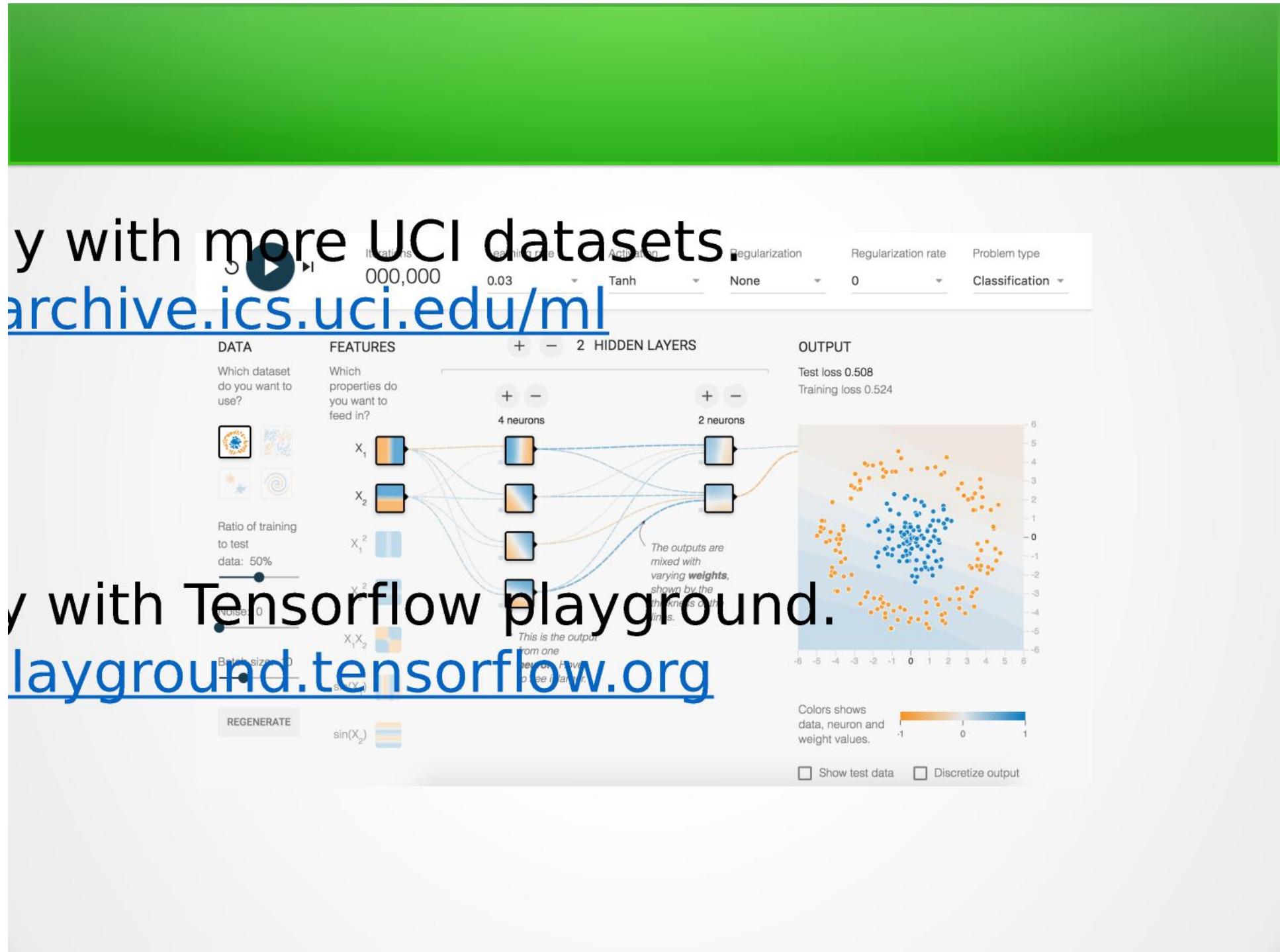
https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
Thorsten Joachims, Cornell University.

An implementation of Support Vector Machines (SVMs) in C.

wpal Wabbit

Microsoft Research and (previously) Yahoo! Research
Fast and scalable tool for learning linear model.

- **Mahout on Hadoop.**
- **MILib on Spark.**
- **Petuum.**



https://en.wikipedia.org/wiki/Hyperparameter_optimization
<http://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/>