Inteligencia Artificial - IAI84

Instituto Tecnológico Metropolitano

Pedro Atencio Ortiz - 2018

En este notebook se aborda el tema de aprendizaje de máquina para clasificación binario utilizando k-Vecinos Cercanos:

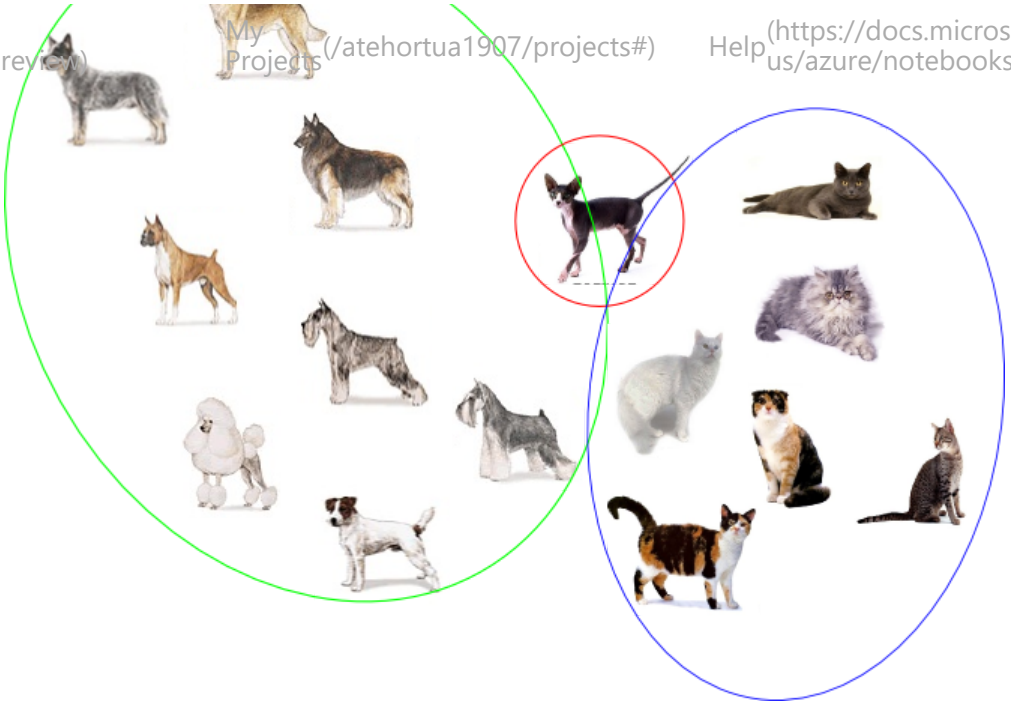
- Clasifidores por distancia o cercanía
- Medidas de distancia o similitud
- Clasificación según vecino más próximo
- Clasificación según k vecinos más próximos (k-NN)

Módulo 4: Machine Learning

Módulo 4_2: Aproximaciones Tradicionales

Módulo 4_2_1: Clasificación por distancia (k-NN)

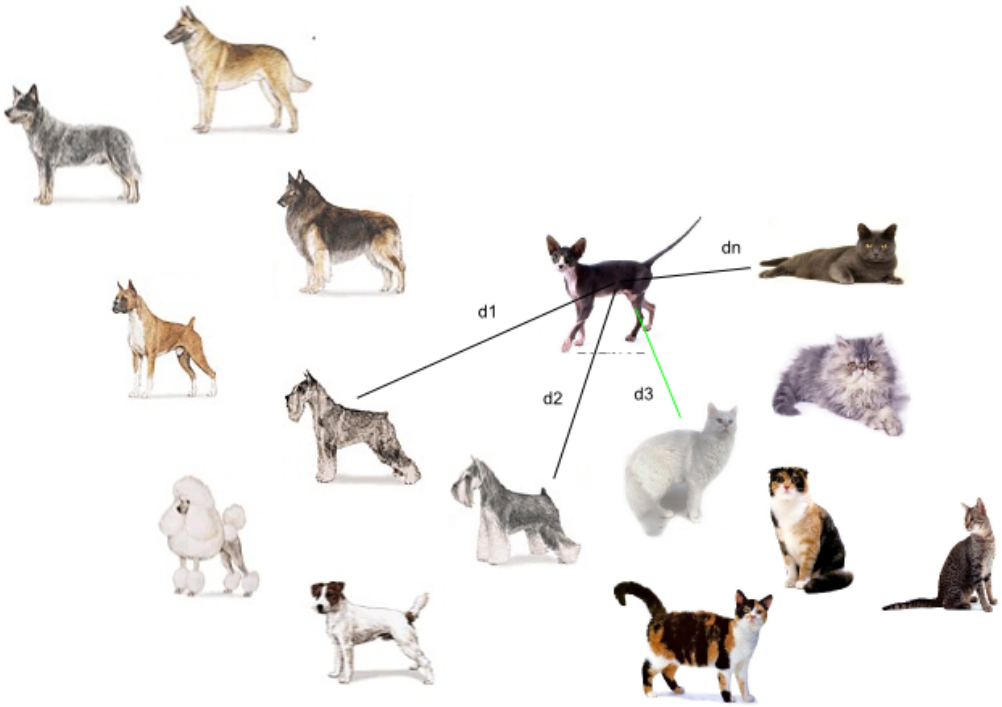
- Estos algoritmos se basan en la suposición de que las muestras pertenecientes a una misma clase, estarán muy próximas entre sí en el espacio de representación.
- Esta suposición implica contar con alguna medida de similitud entre datos de las clases.
- Esto supone de igual forma que la mayoría de los datos de entrenamiento están clasificados de forma correcta.



Estos clasificadores también suelen ser conocidos como:

- Clasificadores basados en memoria.
- Aprendizaje perezoso (lazy learning).
- Clasificadores basados en instancias.

¿cuál es el vecino más próximo?



¿cómo medimos la proximidad?

Medida de proximidad

Una medida comúnmente utilizada para k-NN es la distancia euclídea:

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum (x^{(i)} - x^{(j)})^2}$$

Donde $x^{(i)}$ y $x^{(j)}$ son dos ejemplos del dataset.

In [3]:
```
import numpy as np
import time
import matplotlib.pyplot as plt
```

In [4]:
```
def vect_euclidean_dist(x_i, x_j):
    """
    Implements a euclidean distance between two arrays.
    Arrays must be two-dimensional

    Arguments:
    x_i -- array i
    x_j -- array j (can be a matrix)

    Returns:
    euclidean distance
    """
    return np.sqrt(np.sum((x_i - x_j)**2, axis=1))
```

In [5]:
```
x_1 = np.array([[1.3,2.2,3.1]])
x_2 = np.array([[1.4,2.5,3.8], [1.4,2.0,3.8], [1.4,-6,3.8]])

print(vect_euclidean_dist(x_1, x_2))
```

```
[0.76811457 0.73484692 8.23043134]
```

Clasificación por vecino más próximo

El algoritmo del vecino más próximo consiste entonces en clasificar un objeto con la misma clase de otro objeto con el cual tenga una menor distancia. Es decir, la clase que tomará un objeto nuevo, consiste en la misma clase del objeto más similar a este respecto a sus características.

Para ello entonces obtenenemos las distancias del objeto nuevo respecto a todos los objetos del dataset y seleccionamos la categoría del objeto más cercano.

In [6]:
```
x_i = np.array([[8.8, 7.5]])
```

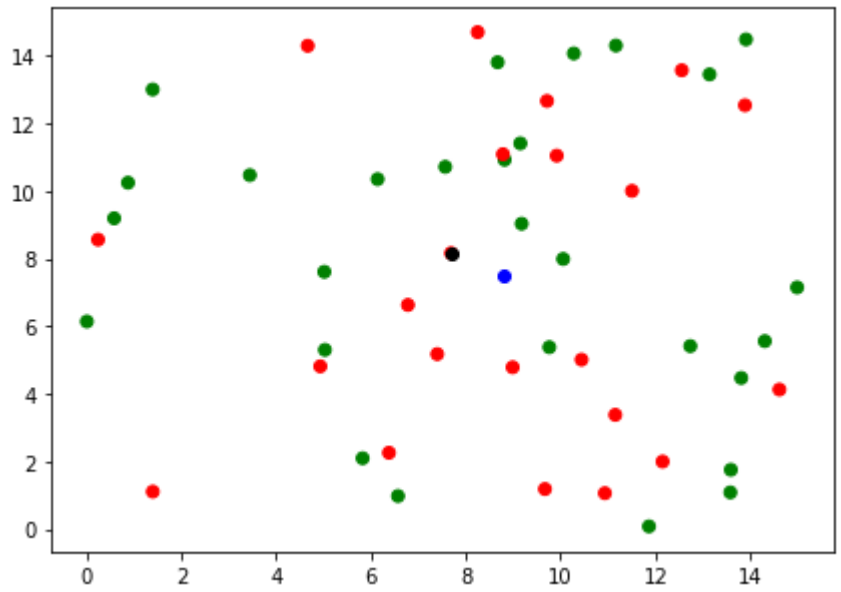
```
X = np.random.rand(50, 2) * 15
Y = np.round(np.random.rand(50, 1))
```

In [7]:
```
def nearest_neighbor(x_i, X):
    min_dist = 1000000
    n_index = 0

    for i in range(X.shape[0]):
        temp_dist = vect_euclidean_dist(x_i, X[i,:])
        if(temp_dist < min_dist):
            min_dist = temp_dist
            n_index = i

    return n_index
```

In [8]:
```
nearest = nearest_neighbor(x_i, X)
print("class: ", np.squeeze(Y[nearest]))

print("nearest neighbor: ", X[nearest])
```

```
('class: ', array(1.0))
('nearest neighbor: ', array([ 7.68915106,  8.16324632]))
```

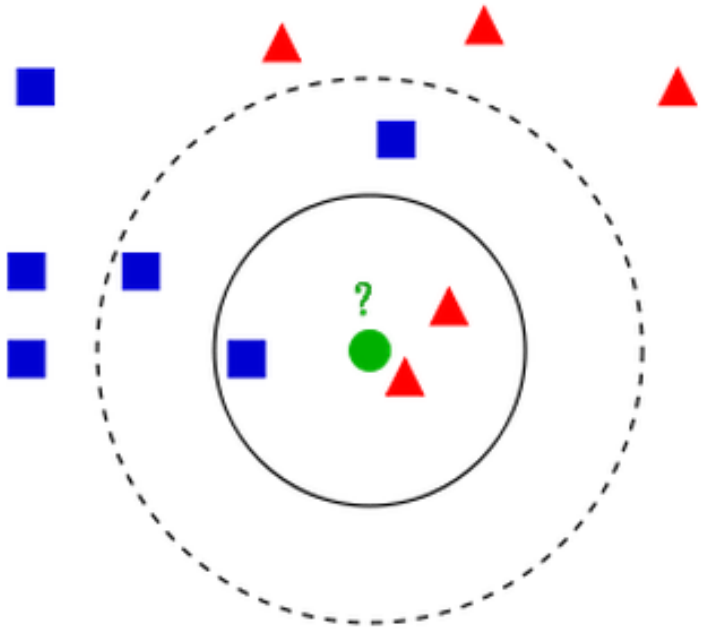
In [12]:
```
import matplotlib

color= ['red' if y == 1 else 'green' for y in Y]

plt.figure(figsize=(7,5))
plt.scatter(X[:,0], X[:,1], color=color)
plt.scatter(x_i[:,0], x_i[:,1], color='blue')
plt.scatter(X[nearest][0], X[nearest][1], color='black')

plt.show()
```



k-Vecinos Cercanos (k-NN)

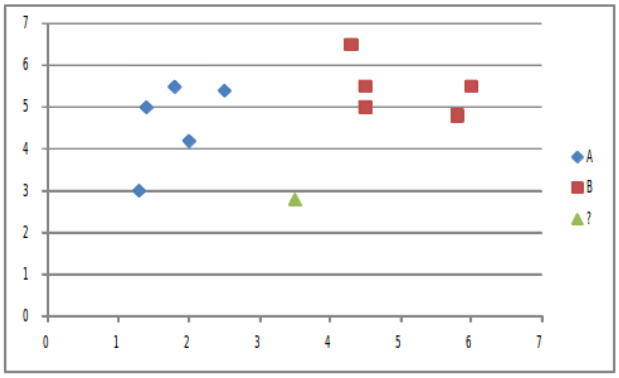
k-vecinos cercanos (k-NN)

Una aproximación más sofisticada, clasificación k-NN, encuentra un grupo de k objetos en el conjunto de entrenamiento que se encuentran más cerca del objeto de prueba, y asigna una clase al mismo basado en la predominancia de una clase particular en el vecindario.



Dados un conjunto de entrenamiento (X, Y) y un objeto de prueba x_i, el algoritmo computa la distancia o similaridad entre x_i y todos los objetos de entrenamiento que pertenecen a (X, Y) para determinar la lista de vecinos más cercanos. Una vez se obtiene dicha lista, x_i se clasifica con la clase de mayor aparición en su vecindario (mayoría de votos).

Indice	X1	X2	Y
1	1,3	3	A
2	1,4	5	A
3	2	4,2	A
4	2,5	5,4	A
5	1,8	5,5	A
6	4,5	5	B
7	5,8	4,8	B
8	4,5	5,5	B
9	6	5,5	B
10	4,3	6,5	B
11	3,5	2,8	?



In [13]:
```
def k_nearest_neighbor(x_i, X, k):
    """
```

```
    Implements a k-NN classifier using euclidean distance.

    Arguments:
    x_i -- array i
    X -- two-dimensional array containing training samples features
    k -- number of neighbors

    Returns:
    array of indexes of nearest neighbors
    """
    distances = vect_euclidean_dist(x_i, X)
    ordered_index = np.argsort(distances)
    nearest_index = ordered_index[0:k]
    return nearest_index
```

In [14]:
```
from collections import Counter

print("nearest neighbors: ", X[nearest])

nearest = k_nearest_neighbor(x_i, X, 11)
print("neighbors classes: ", Y[nearest])

counter = Counter(np.squeeze(Y[nearest]))
print("class: ", counter.most_common(1)[0][0])
```

```
('nearest neighbors: ', array([ 7.68915106,  8.16324632]))
('neighbors classes: ', array([[ 1.],
       [ 0.],
       [ 0.],
       [ 1.],
       [ 0.],
       [ 1.],
       [ 1.],
       [ 1.],
       [ 0.],
       [ 0.],
       [ 1.]]))
('class: ', 1.0)
```

Una implementación real de kNN requiere crear un árbol kd-Tree para que sea posible encontrar los vecinos cercanos en un tiempo razonable computacionalmente, de tal forma que no sea necesario comparar cada nuevo dato respecto a todo el dataset. Para una lectura al respecto ir a: https://ashokharnal.wordpress.com/2015/01/20/a-working-example-of-k-d-tree-formation-and-k-nearest-neighbor-algorithms/ (https://ashokharnal.wordpress.com/2015/01/20/a-working-example-of-k-d-tree-formation-and-k-nearest-neighbor-algorithms/)

SKLEARN

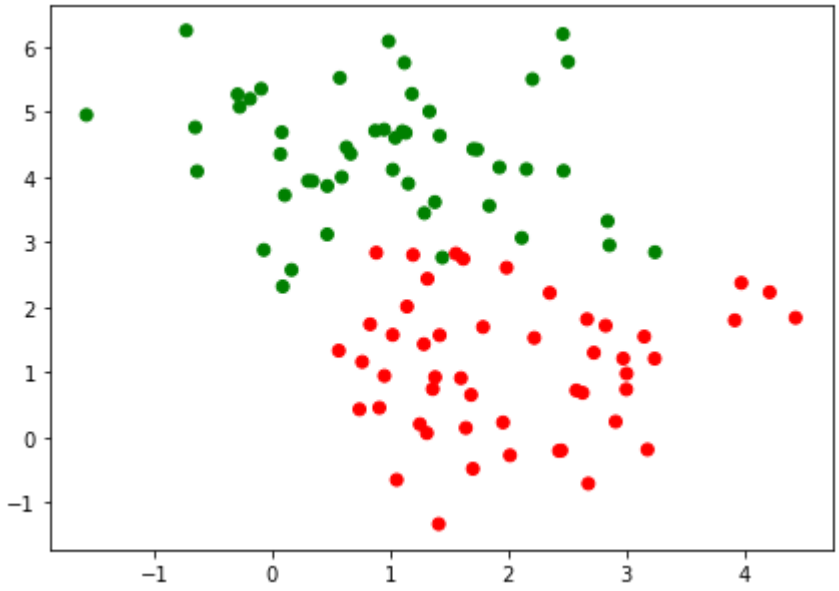
Esta librería tiene implementadas múltiples técnicas de aprendizaje de máquina y de manejo de datasets. Realicemos una prueba simple.

In [15]:
```
'''
    Utility functions
'''

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

def generate_data(data_type):
    """
    Generate a binary dataset with distribution data_type

    Arguments:
    data_type -- distribution of dataset {moons,circles,blobs}

    Returns:
    X -- features
    Y -- labels
    """
    np.random.seed(0)
    if data_type == 'moons':
        X, Y = datasets.make_moons(200, noise=0.20)
    elif data_type == 'circles':
        X, Y = sklearn.datasets.make_circles(200, noise=0.20)
    elif data_type == 'blobs':
        X, Y = sklearn.datasets.make_blobs(centers=2, random_state=0)
    return X, Y

def visualize(X, y, model):
    plot_decision_boundary(lambda x:model.predict(x), X, y)

def plot_decision_boundary(pred_func, X, y):
    # Set min and max values and give it some padding
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_
    # Predict the function value for the whole gid
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.figure(figsize=(7,5))
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
    plt.show()
```

In [16]:
```
X, Y = generate_data('blobs')
```

In [17]:
```
color= ['red' if y == 1 else 'green' for y in Y]
```

```
plt.figure(figsize=(7,5))
plt.scatter(X[:,0], X[:,1], color=color)

plt.show()
```

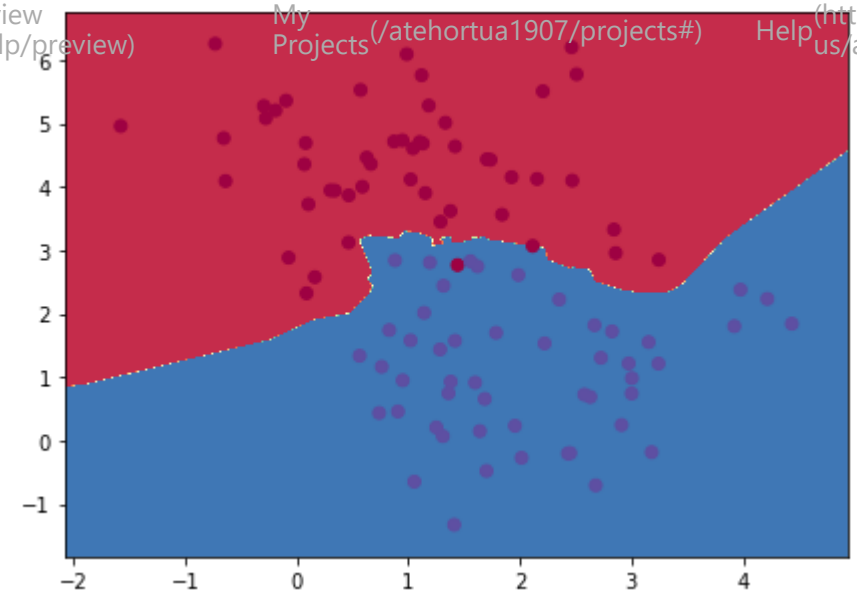


Implementemos kNN con SKLearn

In [29]:
```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
neigh.fit(X, Y)
```

Out[29]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidea
n',
                     metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                     weights='uniform')
```

In [30]:
```
visualize(X, Y, neigh)
```



Trabajemos

- ¿Como se comporta kNN ante distintos valores de k?
- ¿Como se comporta kNN ante datasets de diferentes distribuciones?
- ¿Que pasa cuando utilizamos una porción de los datos para entrenar (X_train, Y_train) y otra para validar (X_dev, Y_dev)?

In [ ]: