# MEMORIA PRÁCTICA 5

Alejandro Santorum - alejandro.santorum@estudiante.uam.es
Inteligencia Artificial
Práctica 5 Pareja 9

10 de mayo de 2019

## Contents

**1 Introducción** **2**

**2 Construcción de clasificadores en bases de datos sintéticas** **2**

**3 Construcción de un clasificador en una base de datos real** **5**

# 1 Introducción

Finalmente llegamos a la última práctica de la asignatura de Inteligencia Artificial. Esta tiene como objetivo iniciarnos en el mundo del aprendizaje automático o *Machine Learning*.

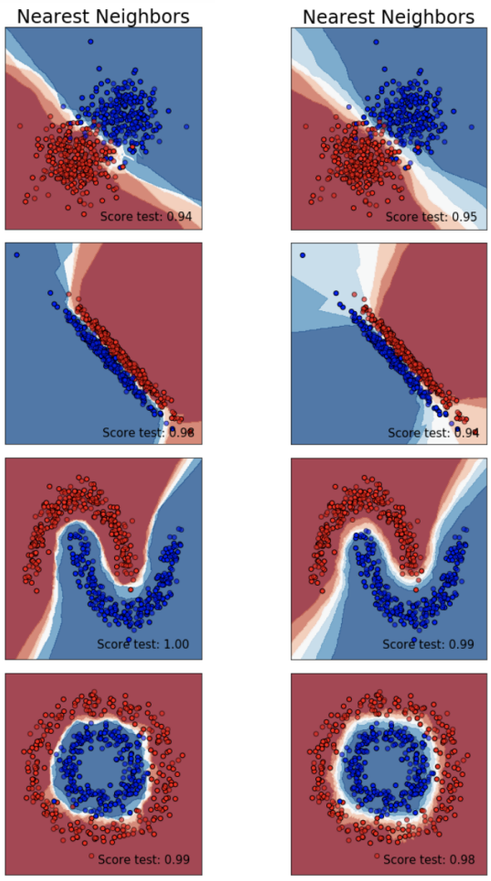
En esta práctica deberemos analizar cómo varía el comportamiento de un algoritmo de aprendizaje automático con la variación en sus parámetros de entrenamiento.

# 2 Construcción de clasificadores en bases de datos sintéticas

En esta sección se manipularan 4 conjuntos de entrenamiento sintéticos con 5 diferentes clasificadores: `Naïve-Bayes`, `K-Vecinos más próximos`, `Árboles de decisión`, `Regresión logística` y `Redes neuronales`.

**Pregunta 1: Varía el número de vecinos de K-NN y contesta: ¿Por qué siempre debe ser impar cuando hay dos clases?**

Variando el parámetro $k$ de K-NN podemos ver que rendimiento tiene con los 4 conjuntos de entrenamiento. Entre $k = 2$ y $k = 11$ el agoritmo después de ser entrenado tiene buena precisión; no obstante, según aumentamos $k$ las fronteras empiezan a tener comportamientos raros, llegando a ser significativo a partir de $k = 20$ (a pesar de tener una precisión aceptable).

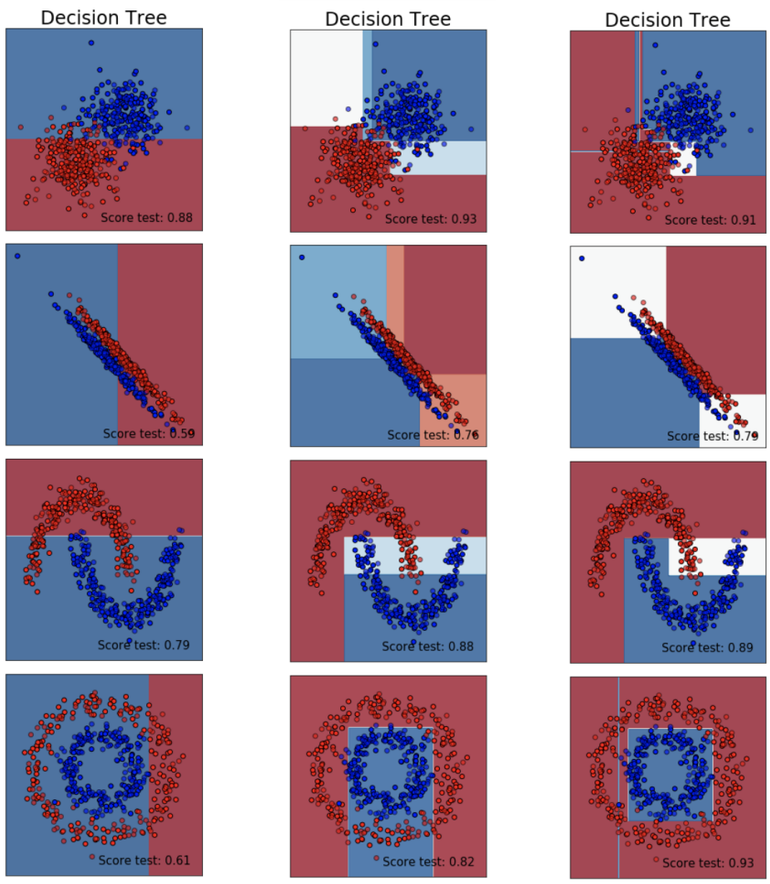



(a) k = 5 (b) k = 21

Figure 1: Comportamiento de la frontera variando k

Como respuesta a la pregunta plateada, el entero $k$ debe ser siempre impar cuando tenemos que decidir entre dos posibles clases para evitar que exista un empate en el recuento de clases de los $k$ vecinos más próximos. En el caso de que para un dato los $k$ vecinos más próximos sean $k/2$ de una clase y $k/2$ de la otra, la clase predicha para ese dato se escogerá aleatoriamente (equivalente a lanzar una moneda al aire). Por esta razón, cuando intentamos clasificar entre dos clases, $k$ debe ser impar para evitar estas situaciones.

**Pregunta 2: Varía la profundidad máxima de los árboles de decisión y comenta como cambia la frontera.**




(a) d = 1 (b) d = 3 (c) d = 4

Figure 2: Comportamiento de la frontera variando la profundidad

Las gráficas mostradas son los resultados obtenidos con el entrenamiento de árboles de decisión con profundidad máxima $d = 1$, $d = 3$ y $d = 4$.

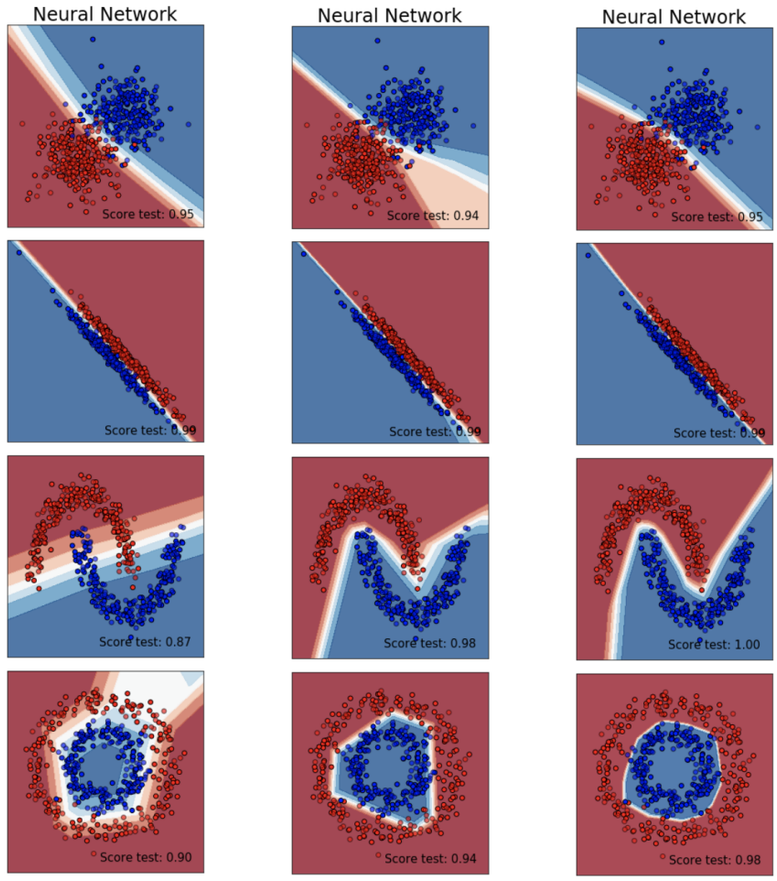
Para $d = 1$ es trivial ver que el modelo está teniendo un rendimiento muy bajo, es decir, está sufriendo de *underfitting*.

Para $d = 3$ alcanzamos el mejor rendimiento de este modelo con estos datos. NO es una maravilla pero sí que es mejor que $d = 1$ o $d = 2$.

En la última imagen ($d = 4$) ya podemos empezar a ver que el modelo está sobreajustando los datos, sobre todo en las imágenes superior e inferior. El modelo sufre de *overfitting*.

**<u>Pregunta 3:</u> Varía el número de capas ocultas y el número de neuronas por capa de las redes neuronales y comenta como cambia la frontera.**

En las imágenes que se mostrarán a continuación podremos ver el comportamiento de las redes neuronales.




(a) 1 capa con 5 neuronas

(b) 2 capas con 5 neuronas

(c) 3 capas con 15 neuronas

Figure 3: Comportamiento de la frontera variando #capas & #neuronas

Con una capa de cinco neuronas podemos ver que solo se ajusta correctamente el segundo conjunto de entrenamiento. También se observa que en el tercer y cuarto conjunto de entrenamiento la red neuronal está sufriendo de *underfitting*.

Con dos capas de cinco neuronas cada una podemos ver una clara mejora de los resultados. No obstante las fronteras se ven forzadas, con picos e irregularidades. Posiblemente aumentar el número de neuronas y/o capas puede ayudar a mejorar el rendimiento.

Efectivamente, como vemos en las imágenes de la red neuronal de tres capas con quince neuronas cada una, los resultados son mucho mejores. Las fronteras se notan más suaves y los porcentajes de precisión han aumentado.

Posiblemente aumentar el número de capas o neuronas es contraproducente ahora debido a que tiene pinta que hemos alcanzado ya un rendimiento.

# 3 Construcción de un clasificador en una base de datos real

Ahora se utilizarán datos reales y por lo tanto, con muchos más sesgos que los datos sintéticos. En este caso, se corresponden con medidas y datos médicos y tenemos el objetivo de determinar si un paciente tiene o no diabetes a partir de ellos.

**Pregunta 1: ¿Cuál es el mejor score que consigues con un K-NN y con qué k (valor de n_neighbours)?.**

Para responder a esta pregunta hemos entrenado un modelo de K-NN y lo hemos evaluado en varias particiones training-test de los datos.

Programa para variación del parámetro k

```
for k in range(1, 51):
    clf = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(clf, X, y, cv=5)
    #print("scores: ", scores)
    print(str(k)+": "+"Score global del modelo: {:.2f} +/- {:.2f}".format(
        scores.mean(), scores.std()))
```

Resultados

```
1: Score global del modelo: 0.66 +/- 0.05
2: Score global del modelo: 0.69 +/- 0.02
3: Score global del modelo: 0.71 +/- 0.03
4: Score global del modelo: 0.72 +/- 0.01
5: Score global del modelo: 0.72 +/- 0.02
6: Score global del modelo: 0.73 +/- 0.02
7: Score global del modelo: 0.74 +/- 0.02
8: Score global del modelo: 0.75 +/- 0.02
9: Score global del modelo: 0.74 +/- 0.03
10: Score global del modelo: 0.74 +/- 0.03
11: Score global del modelo: 0.75 +/- 0.03
12: Score global del modelo: 0.75 +/- 0.03
13: Score global del modelo: 0.76 +/- 0.04
14: Score global del modelo: 0.76 +/- 0.03
15: Score global del modelo: 0.74 +/- 0.04
16: Score global del modelo: 0.74 +/- 0.03
17: Score global del modelo: 0.75 +/- 0.05
18: Score global del modelo: 0.75 +/- 0.04
19: Score global del modelo: 0.75 +/- 0.05
20: Score global del modelo: 0.75 +/- 0.04
21: Score global del modelo: 0.74 +/- 0.04
22: Score global del modelo: 0.75 +/- 0.04
23: Score global del modelo: 0.74 +/- 0.03
24: Score global del modelo: 0.74 +/- 0.04
25: Score global del modelo: 0.74 +/- 0.05
26: Score global del modelo: 0.74 +/- 0.03
27: Score global del modelo: 0.74 +/- 0.04
28: Score global del modelo: 0.74 +/- 0.02
29: Score global del modelo: 0.74 +/- 0.03
30: Score global del modelo: 0.75 +/- 0.02
31: Score global del modelo: 0.74 +/- 0.02
32: Score global del modelo: 0.75 +/- 0.02
33: Score global del modelo: 0.75 +/- 0.01
34: Score global del modelo: 0.75 +/- 0.02
35: Score global del modelo: 0.75 +/- 0.01
36: Score global del modelo: 0.75 +/- 0.01
37: Score global del modelo: 0.74 +/- 0.02
38: Score global del modelo: 0.74 +/- 0.02
39: Score global del modelo: 0.74 +/- 0.02
40: Score global del modelo: 0.74 +/- 0.01
41: Score global del modelo: 0.74 +/- 0.02
42: Score global del modelo: 0.73 +/- 0.01
43: Score global del modelo: 0.73 +/- 0.01
44: Score global del modelo: 0.73 +/- 0.01
45: Score global del modelo: 0.74 +/- 0.02
46: Score global del modelo: 0.74 +/- 0.02
```

```
47: Score global del modelo: 0.74 +/- 0.02
48: Score global del modelo: 0.74 +/- 0.02
49: Score global del modelo: 0.74 +/- 0.02
50: Score global del modelo: 0.74 +/- 0.02
```

Con estos resultados podemos dudar qué parámetros son los mejores para $k = 11$, $k = 12$ y $k = 13$. Sin embargo, está claro que entre estos estaría nuestra elección. Posiblemente **escogeríamos k=11**, por ser un número menor y necesitar menos tiempo de entrenamiento y, además, ser **impar** (recordar la respuesta a la pregunta 1 de la primera sección).

**Pregunta 2: ¿Cuál es el mejor score que consigues con un árbol de decisión y con qué profundidad máxima (valor de max_depth)?.**

Para responder a esta pregunta hemos entrenado un modelo de árboles de decisión y lo hemos evaluado en varias particiones training-test de los datos.

Programa para variación del parámetro max_depth

```
for d in range(1, 26):
    clf = DecisionTreeClassifier(max_depth=d)
    scores = cross_val_score(clf, X, y, cv=5)
    #print("scores: ", scores)
    print(str(d)+": "+"Score global del modelo: {:.2f} +/- {:.2f}".format(
        scores.mean(), scores.std()))
```

Resultados

```
1: Score global del modelo: 0.72 +/- 0.03
2: Score global del modelo: 0.74 +/- 0.01
3: Score global del modelo: 0.73 +/- 0.01
4: Score global del modelo: 0.73 +/- 0.03
5: Score global del modelo: 0.75 +/- 0.04
6: Score global del modelo: 0.72 +/- 0.04
7: Score global del modelo: 0.73 +/- 0.02
8: Score global del modelo: 0.71 +/- 0.02
9: Score global del modelo: 0.72 +/- 0.03
10: Score global del modelo: 0.72 +/- 0.05
11: Score global del modelo: 0.72 +/- 0.04
12: Score global del modelo: 0.71 +/- 0.05
13: Score global del modelo: 0.72 +/- 0.04
14: Score global del modelo: 0.72 +/- 0.06
15: Score global del modelo: 0.72 +/- 0.04
16: Score global del modelo: 0.71 +/- 0.04
17: Score global del modelo: 0.71 +/- 0.04
18: Score global del modelo: 0.71 +/- 0.04
19: Score global del modelo: 0.71 +/- 0.04
20: Score global del modelo: 0.71 +/- 0.05
21: Score global del modelo: 0.71 +/- 0.04
22: Score global del modelo: 0.70 +/- 0.04
23: Score global del modelo: 0.73 +/- 0.04
24: Score global del modelo: 0.72 +/- 0.05
25: Score global del modelo: 0.70 +/- 0.04
```

En este caso la respuesta, viendo los resultados, es claro: las mejores predicciones se obtienen con una **profundad máxima igual a 2**.

**Pregunta 3: ¿Cuál es el mejor score que consigues con una red neuronal y con qué configuración (valor de hidden_layer_sizes)?.**

Finalmente, para las redes neuronales hemos realizado el mismo proceso que en los dos casos anteriores:

Programa para variación del #capas ocultas y #neuronas

```
for nlayers in range(1,6):
    for nneurons in range(5, 106, 10):
        hid_lay_array = []
        for i in range(nlayers+1):
            hid_lay_array.append(nneurons)
        hid_lay_array = tuple(hid_lay_array)
        clf = MLPClassifier(hidden_layer_sizes=hid_lay_array, max_iter
           =1000,alpha=0)
        scores = cross_val_score(clf, X, y, cv=5)
        print("NLayers:"+str(nlayers)+" NNeurons:"+str(nneurons)+" ->
           Score global del modelo: {:.2f} +/- {:.2f}".format(scores.mean
           (), scores.std()))
```

Resultados

```
NLayers:1 NNeurons:5 -> Score global del modelo: 0.67 +/- 0.02
NLayers:1 NNeurons:15 -> Score global del modelo: 0.67 +/- 0.04
NLayers:1 NNeurons:25 -> Score global del modelo: 0.68 +/- 0.02
NLayers:1 NNeurons:35 -> Score global del modelo: 0.67 +/- 0.02
NLayers:1 NNeurons:45 -> Score global del modelo: 0.67 +/- 0.03
NLayers:1 NNeurons:55 -> Score global del modelo: 0.68 +/- 0.04
NLayers:1 NNeurons:65 -> Score global del modelo: 0.69 +/- 0.02
NLayers:1 NNeurons:75 -> Score global del modelo: 0.69 +/- 0.05
NLayers:1 NNeurons:85 -> Score global del modelo: 0.66 +/- 0.04
NLayers:1 NNeurons:95 -> Score global del modelo: 0.69 +/- 0.02
NLayers:1 NNeurons:105 -> Score global del modelo: 0.69 +/- 0.01
NLayers:2 NNeurons:5 -> Score global del modelo: 0.69 +/- 0.01
NLayers:2 NNeurons:15 -> Score global del modelo: 0.71 +/- 0.02
NLayers:2 NNeurons:25 -> Score global del modelo: 0.68 +/- 0.02
NLayers:2 NNeurons:35 -> Score global del modelo: 0.70 +/- 0.03
NLayers:2 NNeurons:45 -> Score global del modelo: 0.69 +/- 0.03
NLayers:2 NNeurons:55 -> Score global del modelo: 0.69 +/- 0.05
NLayers:2 NNeurons:65 -> Score global del modelo: 0.67 +/- 0.03
NLayers:2 NNeurons:75 -> Score global del modelo: 0.65 +/- 0.07
NLayers:2 NNeurons:85 -> Score global del modelo: 0.71 +/- 0.04
NLayers:2 NNeurons:95 -> Score global del modelo: 0.67 +/- 0.04
NLayers:2 NNeurons:105 -> Score global del modelo: 0.65 +/- 0.05
NLayers:3 NNeurons:5 -> Score global del modelo: 0.65 +/- 0.00
NLayers:3 NNeurons:15 -> Score global del modelo: 0.71 +/- 0.04
NLayers:3 NNeurons:25 -> Score global del modelo: 0.68 +/- 0.04
NLayers:3 NNeurons:35 -> Score global del modelo: 0.67 +/- 0.05
NLayers:3 NNeurons:45 -> Score global del modelo: 0.68 +/- 0.03
NLayers:3 NNeurons:55 -> Score global del modelo: 0.69 +/- 0.03
NLayers:3 NNeurons:65 -> Score global del modelo: 0.66 +/- 0.04
NLayers:3 NNeurons:75 -> Score global del modelo: 0.66 +/- 0.05
NLayers:3 NNeurons:85 -> Score global del modelo: 0.69 +/- 0.04
NLayers:3 NNeurons:95 -> Score global del modelo: 0.70 +/- 0.01
NLayers:3 NNeurons:105 -> Score global del modelo: 0.68 +/- 0.04
NLayers:4 NNeurons:5 -> Score global del modelo: 0.66 +/- 0.04
NLayers:4 NNeurons:15 -> Score global del modelo: 0.69 +/- 0.04
NLayers:4 NNeurons:25 -> Score global del modelo: 0.68 +/- 0.05
NLayers:4 NNeurons:35 -> Score global del modelo: 0.69 +/- 0.04
NLayers:4 NNeurons:45 -> Score global del modelo: 0.70 +/- 0.02
NLayers:4 NNeurons:55 -> Score global del modelo: 0.68 +/- 0.03
NLayers:4 NNeurons:65 -> Score global del modelo: 0.69 +/- 0.02
NLayers:4 NNeurons:75 -> Score global del modelo: 0.67 +/- 0.04
NLayers:4 NNeurons:85 -> Score global del modelo: 0.67 +/- 0.03
NLayers:4 NNeurons:95 -> Score global del modelo: 0.68 +/- 0.03
NLayers:4 NNeurons:105 -> Score global del modelo: 0.70 +/- 0.03
NLayers:5 NNeurons:5 -> Score global del modelo: 0.68 +/- 0.04
NLayers:5 NNeurons:15 -> Score global del modelo: 0.71 +/- 0.03
NLayers:5 NNeurons:25 -> Score global del modelo: 0.68 +/- 0.02
NLayers:5 NNeurons:35 -> Score global del modelo: 0.71 +/- 0.02
NLayers:5 NNeurons:45 -> Score global del modelo: 0.69 +/- 0.03
```

```
NLayers:5 NNeurons:55 -> Score global del modelo: 0.69 +/- 0.01
NLayers:5 NNeurons:65 -> Score global del modelo: 0.69 +/- 0.01
NLayers:5 NNeurons:75 -> Score global del modelo: 0.67 +/- 0.02
NLayers:5 NNeurons:85 -> Score global del modelo: 0.70 +/- 0.03
NLayers:5 NNeurons:95 -> Score global del modelo: 0.69 +/- 0.01
NLayers:5 NNeurons:105 -> Score global del modelo: 0.68 +/- 0.05
```

Se pueden observar resultados muy parejos, pero seguramente el mejor resultado se obtiene con dos capas de 15 neuronas cada una.

Podríamos haber continuado probando nuevos parámetros y nuevas configuraciones, quizá hasta cambiando el parámetro de regularización, la función de activación o incluso el algoritmo de optimización, pero no se espera que los resultados cambien significativamente.