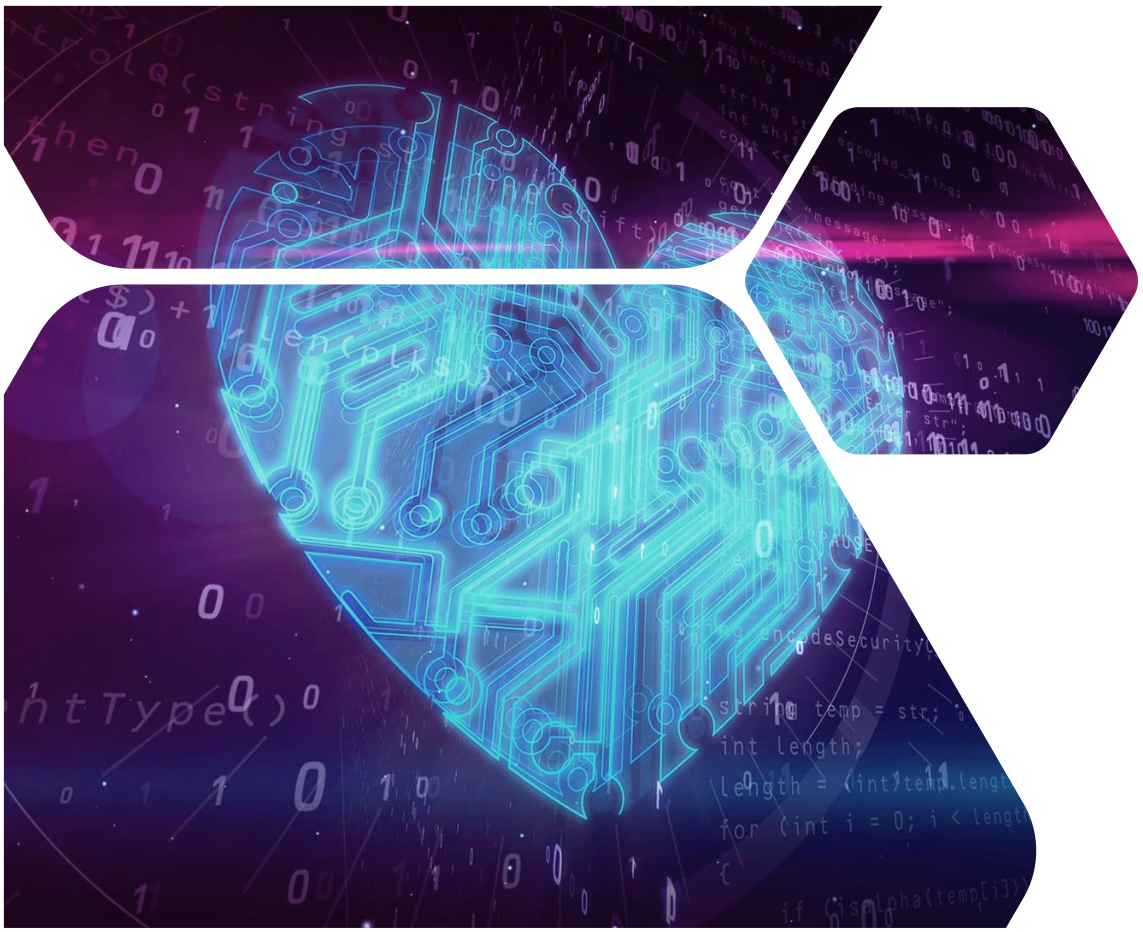
SISTEMAS DE APRENDIZAJE AUTOMÁTICO

Técnicas de Machine Learning para la Predicción de las Enfermedades Cardíacas

Néstor Cabrero Martín
Marcos García García



Contenido

Introducción

El objetivo de este proyecto es estudiar la aplicación de distintos algoritmos de **Machine Learning** sobre el dataset **Heart Disease UCI**, disponible para descargar en la web de **Kaggle.** Este dataset recoge información sobre algunos de los factores de riesgo implicados en la mayoría de enfermedades cardíacas, por lo que puede ser utilizado para su detección temprana.

En un primer acercamiento al dataset, intentaremos entender cada uno de los atributos que presenta, así como tratar los datos de modo que sean más claros para su visualización. Por último, veremos de qué forma podemos tratar los datos para ajustarlos a nuestros modelos sin que se produzcan resultados inesperados.

Una vez estudiados y tratados los datos, aplicaremos tres algoritmos de clasificación de Machine Learning para comprobar la precisión de nuestros modelos: **regresión logística**, **redes neuronales** y **support vector machines** (SVM). En primera instancia utilizaremos las implementaciones que hemos desarrollado a lo largo del curso, ajustando los parámetros necesarios para observar posibles diferencias en los resultados. Posteriormente, utilizaremos algunos módulos de **SKLearn** para comparar nuestra aproximación con la de una librería profesional.

Descripción de los datos

La dataset estudiada presenta una serie de parámetros interesantes para su estudio. Para trabajarlos, hemos decidido realizar una traducción al español de estos. Son los siguientes:

- **age:** representa la edad de los pacientes. Lo hemos traducido como “edad”.
- **sex**: representa el sexo de los pacientes. Lo hemos traducido como “sexo”.
    - Valor 0: mujer.
    - Valor 1: hombre.
- **cp:** representa el tipo de dolor que sienten los pacientes. Traducido como “dolor”.
    - Valor 0: sin síntomas.
    - Valor 1: angina atípica.
    - Valor 2: dolor no anginal.
    - Valor 3. angina típica.
- **trestbps:** representa la tensión arterial de los pacientes. Traducido como “tension”.
- **chol:** es el nivel de colesterol en sangre. Lo hemos traducido como “colesterol”.
- **fbs:** representa la glucemia en ayunas, es decir, la presencia de azúcar en la sangre de los pacientes. Lo hemos traducido como “glucemia”.
    - Valor 0: no excede los 120 mg/ml. Se trata de un valor normal.
    - Valor 1: excede los 120 mg/ml. Se trata de un valor excesivo.
- **restecg:** representa los resultados del electrocardiograma en reposo. Lo hemos traducido a “ecg”.
    - Valor 0: normal.
    - Valor 1: el paciente presenta una anomalía ST-T.
    - Valor 2: el paciente presenta signos de una probable hipertrofia VI.
- **thalach:** representa el máximo ritmo cardiaco presentado por el paciente. Lo hemos traducido como “frec_cardiaca”.
- **exang:** representa si el paciente presenta una angina inducida por la realización de ejercicio. Lo hemos traducido a “angina_ej”.
    - Valor 0: no la presenta.
    - Valor 1: sí la presenta.
- **oldpeak:** representa la depresión de ST inducida por el deporte relativa al reposo. Lo hemos traducido como “depresion_st”.
- **slope:** representa la pendiente del segmento ST. Traducido como “pendiente_st”.
    - Valor 0: decreciente.
    - Valor 1: plana.
    - Valor 2: creciente.
- **ca:** representa el número de vasos mayores del paciente. Puede ser de 0 a 3. Lo hemos traducido como “num_valvulas”.
- **thal:** representa el resultado del test talio, una prueba de esfuerzo en la que se usa material radioactivo (talio) para mostrar qué tan bien fluye la sangre hacia el músculo cardíaco, tanto en reposo como en actividad. Lo hemos traducido como “test_talio”.
    - Valor 1: presenta un defecto permanente.
    - Valor 2: normal.
    - Valor 3: presenta un defecto reversible.
- **target:** representa si el paciente padece una enfermedad cardiaca. Lo hemos traducido como “enfermedad”.
    - Valor 0: no la padece.
    - Valor 1: sí la padece.

Utilizando la librería de **Pandas Profiling** podemos obtener un informe altamente detallado del dataset y cada una de sus características, con datos estadísticos y de interés para la aplicación de nuestros algoritmos. El informe completo puede consultarse en los anexos; este es el resumen general:

Dataset info

Number of variables	14
Number of observations	303
Missing cells	0 (0.0%)
Duplicate rows	1 (0.3%)
Total size in memory	33.2 KiB
Average record size in memory	112.3 B

Variables types

NUM	6
CAT	4
BOOL	4

Visualización y Tratamiento de los datos

De entre todas las aplicaciones que pueden beneficiarse de técnicas de Machine Learning, las diseñadas para el **diagnóstico de enfermedades graves** son quizás las menos indicadas para ser cajas negras. Si los valores de salida de un determinado modelo cambian el tratamiento de un paciente (con los potenciales efectos secundarios que eso conlleva), o indican la necesidad de cirugía, o la suspensión de un tratamiento, los pacientes querrán saber por qué.

Este dataset nos proporciona una serie de variables que pueden o no condicionar el tener o no una enfermedad cardíaca. Es por ello que parece importante discernir qué variables se tomarán más en cuenta, y tratar de entender cómo afecta cada una de ellas al resultado final. En primer lugar, hemos utilizado un algoritmo de **bosque aleatorio**, para luego aplicar diferentes **herramientas y técnicas de explicabilidad**.

En la actualidad, el diagnóstico de la enfermedad cardíaca se realiza a través de la observación de signos y pruebas clínicas. En base al criterio del facultativo, pueden ordenarse distintas pruebas como el electrocardiograma, la tomografía axial computerizada (TAC), el análisis de sangre o pruebas de estrés durante el ejercicio.

Algunos de los factores de riesgo que más se citan en la literatura son un elevado nivel de colesterol, una tensión arterial elevada, diabetes, sobrepeso, adicción al tabaco y factores familiares; otros factores de riesgo que suelen citarse son la edad o el sexo masculino. Cabría esperar, pues, que estos elementos tuviesen una mayor influencia en los datos de salida.

Tras cargar el dataset, podemos visualizar los primeros diez elementos del mismo; es evidente que para visualizarlos adecuadamente vamos a tener que realizar ciertas modificaciones:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

Tal y como indicábamos en el apartado anterior, hemos traducido los nombres de las características de la muestra. Además, hemos alterado los valores numéricos (y su tipo de datos) para que las visualizaciones sean más representativas:

sexo = 0	sexo_mujer
sexo = 1	sexo_varon
dolor = 1	dolor_asintomatico
dolor = 2	dolor_angina atipica
dolor = 3	dolor_no anginoso
dolor = 4	dolor_angina tipica
glucemia = 0	glucemia_<120mg/ml
glucemia = 1	glucemia_>120mg/ml
ecg = 0	ecg_hipertrofia VI
ecg = 1	ecg_normal
ecg = 2	ecg_anomalia ST-T
angina_ej = 0	angina_ej_no
angina_ej = 1	angina_ej_si
pendiente_st = 1	pendiente_st_decreciente
pendiente_st = 2	pendiente_st_plana
pendiente_st = 3	pendiente_st_creciente
test_talio = 1	test_talio_defecto permanente
test_talio = 2	test_talio_normal
test_talio = 3	test_talio_defecto reversible

Una vez tratados así los datos, podemos utilizar la función *RandomForestClassifier* de **SKLearn** para definir nuestro modelo. Habiendo separado previamente los datos en un conjunto de entrenamiento y uno de prueba, podemos entrenar el modelo con el conjunto de entrenamiento. Al realizar una predicción con el conjunto de prueba (un 20% de los datos de entrada), obtenemos los siguientes valores para la **matriz de confusión**:

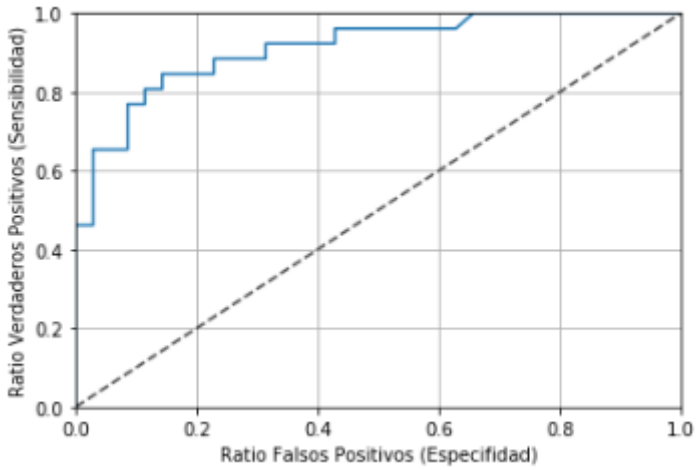
		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- **VP** es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo: **26**
- **VN** es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo: **23**
- **FN** es la cantidad de positivos que fueron clasificados incorrectamente como negativos por el modelo: **9**
- **FP** es la cantidad de negativos que fueron clasificados incorrectamente como positivos por el modelo: **3**

La **sensibilidad** de la clasificación se expresa como el porcentaje de verdaderos positivos sobre el total de positivos. La **especifidad** de la clasificación se expresa como el porcentaje de verdaderos negativos sobre el total de negativos. La **precisión** nos habla de cuántos elementos se han clasificado adecuadamente en relación al total de la muestra.

$$\text{Sensibilidad} = \frac{VP}{\text{Total Positivos}} \quad \text{Especificidad} = \frac{VN}{\text{Total Negativos}}$$

En nuestro modelo, hemos obtenido una sensibilidad del 89’65%, y una especifidad del 71’87%, lo cual está bastante bien. La precisión del modelo es del 80’32%. Para visualizar estos datos utilizamos una **curva ROC** (Receiver Operating Characteristic, representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario):



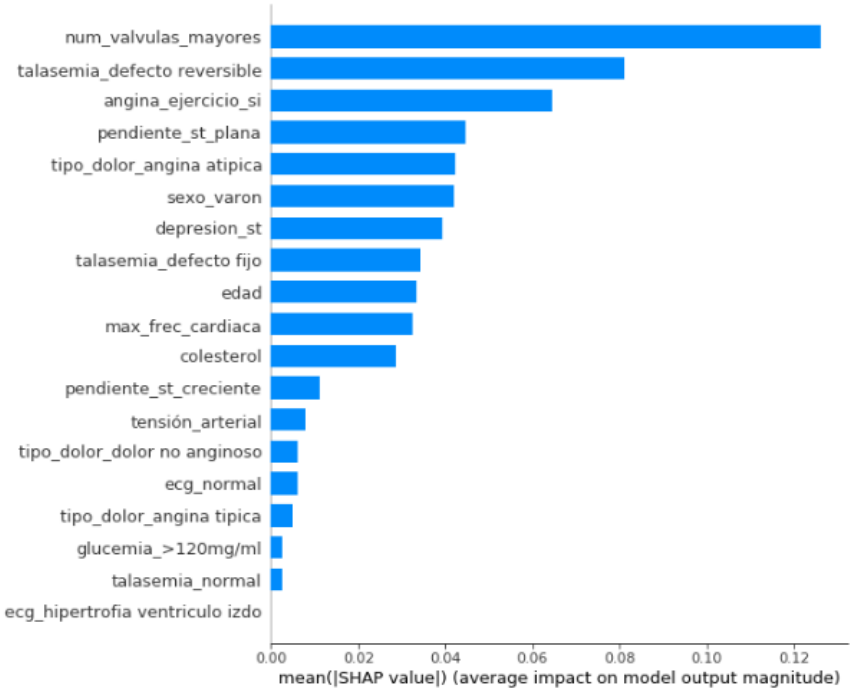
Una buena forma de capturar toda esta información en un solo dato es utilizando la métrica **AUC** (Area Under the Curve); utilizando los datos de la curva ROC obtenemos un valor del 91’26%, lo cual, por lo general, se considera como excelente.

Permutation Importance es una herramienta de explicabilidad que nos facilita la comprensión del modelo; su funcionamiento se basa en aleatorizar variables de forma individual en el conjunto de validación, después del entrenamiento, para ver qué efectos se producen en la precisión del modelo.

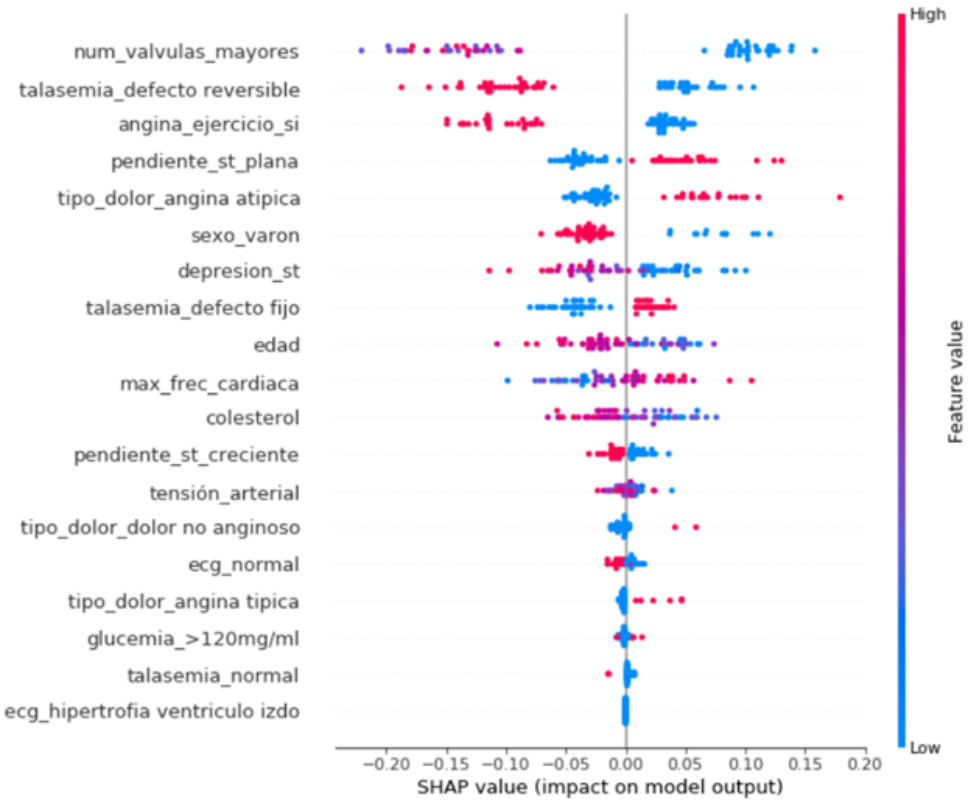
Podemos comprobar que el factor de mayor importancia aquí es el defecto reversible en las pruebas de talio. La elevada importancia de la máxima frecuencia cardíaca tiene sentido, dado que es un factor inmediato sujeto al estado del paciente en el momento de las pruebas (al contrario que, por ejemplo, la edad que es un factor más general)

Weight	Feature
0.0361 ± 0.0321	talasemia_defecto reversible
0.0295 ± 0.0245	sexo_varon
0.0197 ± 0.0382	angina_ejercicio_si
0.0164 ± 0.0000	tipo_dolor_dolor no anginoso
0.0164 ± 0.0464	max_frec_cardiaca
0.0131 ± 0.0245	talasemia_defecto fijo
0.0066 ± 0.0262	depresion_st
0.0033 ± 0.0525	num_valvulas_mayores
0.0033 ± 0.0131	glucemia_>120mg/ml
0.0000 ± 0.0207	tipo_dolor_angina atipica
0 ± 0.0000	talasemia_normal
0 ± 0.0000	tipo_dolor_angina tipica
0 ± 0.0000	ecg_hipertrofia ventriculo izdo
-0.0066 ± 0.0161	ecg_normal
-0.0098 ± 0.0161	pendiente_st_creciente
-0.0131 ± 0.0131	colesterol
-0.0164 ± 0.0207	tensión_arterial
-0.0262 ± 0.0491	edad
-0.0393 ± 0.0445	pendiente_st_plana

SHAP es otra herramienta de explicabilidad que podemos utilizar. En esta gráfica se muestra la influencia de los valores de cada variable en un solo elemento, comparados con sus valores base:

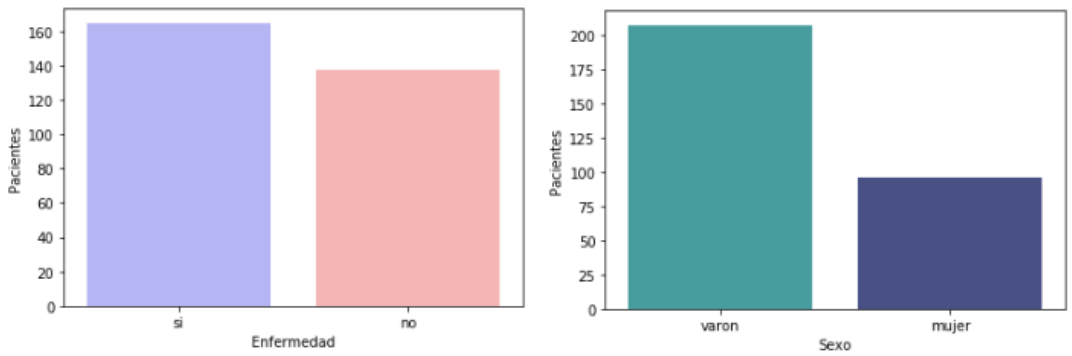


Podemos utilizar el resumen de SHAP para visualizar mejor el impacto que tiene cada característica:

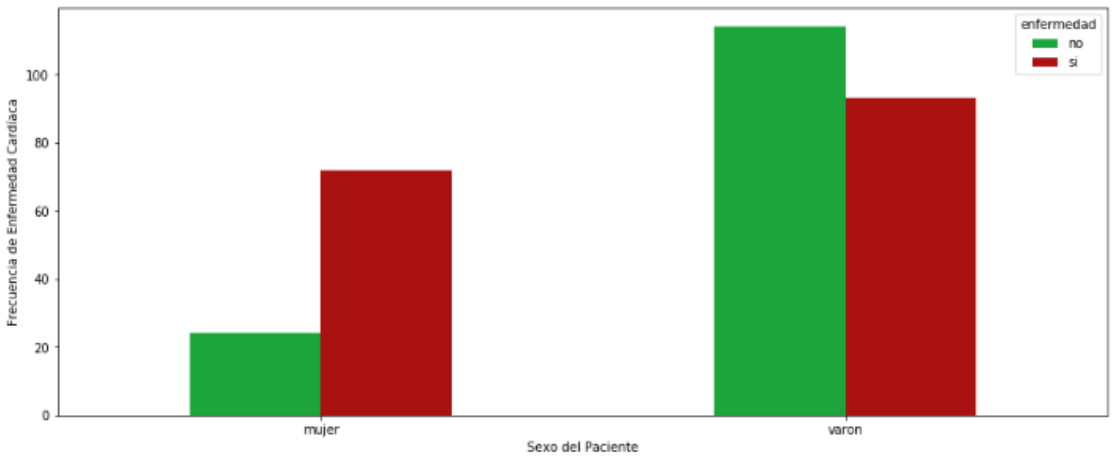


La división en la variable número de válvulas principales es bastante clara, y nos indica que valores bajos son malos. También podemos ver una clara separación en la angina inducida por el ejercicio, cuya presencia es un factor muy fuerte para determinar la presencia de enfermedad cardíaca. Aún así, algunos de los datos no parecen ajustarse a lo que se espera: según el dataset los hombres tendrían menor riesgo de enfermedad cardíaca, o aquellas personas con una curva ST plana, cuando la literatura al respecto dice lo contrario. Aunque el dataset es pequeño, podemos sacar algunas conclusiones, como que los factores más relevantes son los relacionados con las pruebas de ECG o estrés.

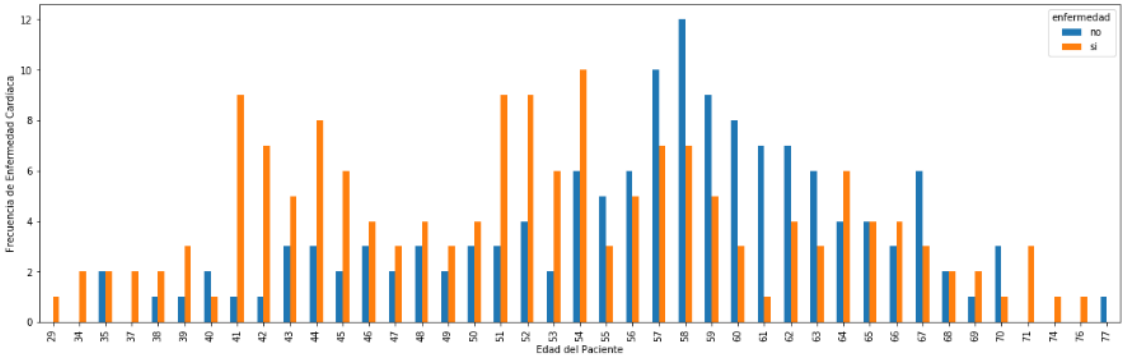
Hasta ahora hemos estudiado cómo se relacionan todas las características del dataset, pero no nos hemos parado a evaluar algunas de las características de forma individual, como, por ejemplo, la relación de pacientes enfermos-no enfermos (54% vs. 45%). Es de notar, por ejemplo, que la mayoría de la muestra está compuesta por varones (más de un 68%), dado que como hemos señalado antes el sexo masculino presenta una incidencia más elevada de enfermedad cardíaca:



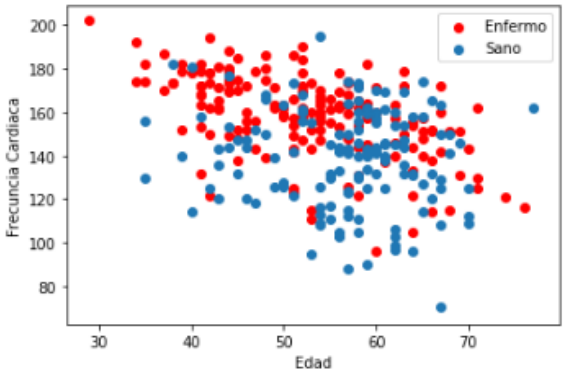
Aun así, se puede observar que el ratio de mujeres enfermas es mucho más elevado que el de los hombres:



Otro de los factores de riesgo que hemos mencionado anteriormente es la edad. La tendencia es que, a mayor edad, mayor es el riesgo de enfermedad cardíaca. Sin embargo, observamos que en la década de los 40 se concentra el mayor riesgo, para luego invertirse la tendencia a partir de la segunda mitad de la década de los 50. Esto es bastante significativo si tenemos en cuenta que la edad media en el dataset está en torno a los 56 años para aquellas personas que no padecen de enfermedad cardíaca, y 52 para aquellas que sí la padecen.

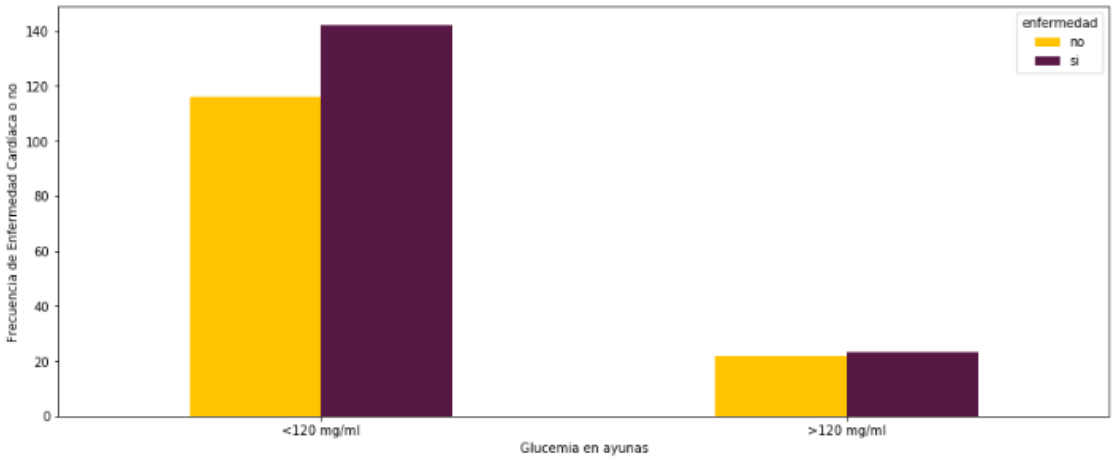


No podemos pasar por alto el hecho de que la edad no es un factor aislado. Por ejemplo, en esta gráfica podemos ver cómo se relaciona con el factor frecuencia cardíaca máxima, que de forma natural desciende con la edad pero que, de algún modo, interviene en la influencia que ejerce el factor edad sobre la enfermedad cardíaca:

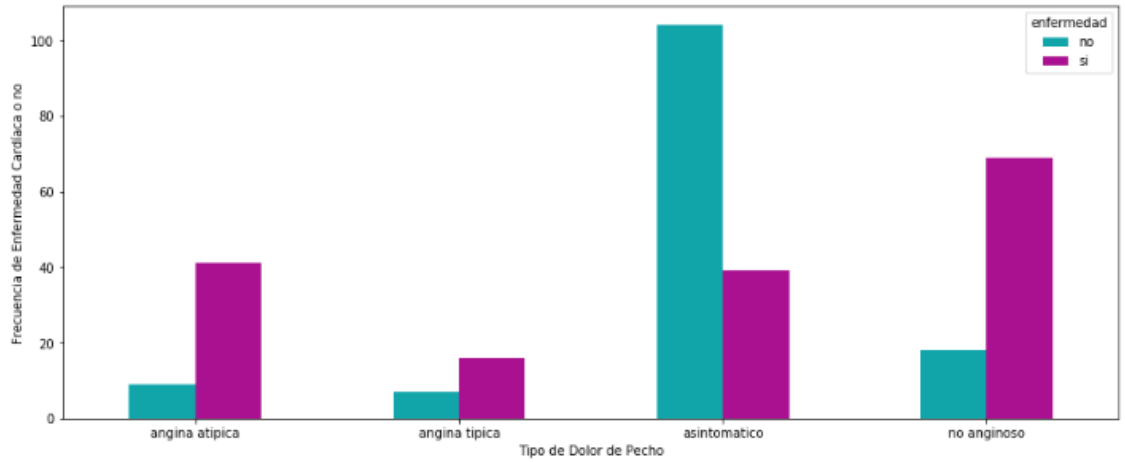


Otro de los factores de riesgo habituales es la glucemia en ayunas. Pese a que está comprobado que una concentración elevada puede llevar a problemas cardíacos, el dataset no revela dicha implicación: por un lado, porque el ratio de pacientes con diabetes no parece muy elevado, y

por otro lado, porque no parece haber mucha diferencia entre el número de pacientes enfermos y no enfermos que, sin embargo, no padecen diabetes. Una vez más, nos encontramos ante un caso de una característica que se ve influenciada por el resto:



El dolor de pecho es un signo clínico evidente de algún tipo de problema cardíaco. Puede resultar interesante, por lo tanto, visualizar cómo afectan sus diferentes tipologías a la aparición o no de enfermedad cardíaca:



En el caso de pacientes asintomáticos observamos lo esperado, puesto que la mayoría no sufren enfermedad cardíaca. Aun así también hay un número elevado de pacientes enfermos, lo que indica una vez más la correlación que hemos venido analizando en los últimos párrafos. Mientras que el dolor anginoso parece reflejar, efectivamente, una angina, los dolores de tipo angina atípica o no anginoso sí que parecen indicar claramente una enfermedad cardíaca.

Como estamos viendo, no es sencillo tratar con tantas variables con unos valores tan dispares. A la hora de aplicar nuestros algoritmos de Machine Learning trataremos los datos, aunque a un nivel inferior que el utilizado hasta ahora.

En primer lugar, prescindiremos de las variables categóricas etiquetadas en el dataset como ‘cp’ (dolor), ‘thal’ (test del talio) y ‘slope’ (pendiente). Para modificar el dataframe de la manera en que esperamos utilizamos la función **get_dummies()** de la librería **Pandas**. Una vez creadas las nuevas categorías, eliminamos aquellas que aparecen duplicadas o sin sentido (las originales ‘cp’, ‘thal’ y ‘slope’). Así, nuestro dataframe pasa a ser de 22 columnas: 21 variables y 1 resultado: enfermo o no enfermo.

Lo primero que haremos tras cargar el archivo 'heart.csv' y acomodar sus columnas será obtener los **valores de entrada** (X) y los **valores de salida** (Y). Antes de realizar cualquier operación sobre los valores de entrada, **normalizamos** los datos; así nos aseguramos unas predicciones fiables a las que, además, podremos aplicar sesgo.

Por último, dividiremos nuestro dataset en un **conjunto de entrenamiento** y en un **conjunto de prueba**. Tal y como hicimos con el modelo del árbol aleatorio, utilizamos la función **train_test_split()** de la librería SKLearn.

Con todos los datos cargados, visualizados y tratados, estamos listos para empezar a aplicar los algoritmos de Machine Learning previstos: regresión logística, redes neuronales y SVM.

Regresión Logística

La **regresión logística** consiste en un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica. Resulta de gran utilidad para modelar la **probabilidad** de un evento ocurriendo como función de otros factores.

Por esto, hemos decidido implementarla para estudiar la probabilidad de que los pacientes padezcan enfermedades cardiacas respecto a dos factores a elegir de entre los parámetros anteriormente listados. De esta forma, se puede estudiar con facilidad cómo afectan dichos parámetros a la hora de padecer la enfermedad o no.

Implementación

En primer lugar, hemos tenido que transformar los parámetros elegidos en la matriz X. Para ello, eliminamos de los datos todos los demás atributos que se queden fuera del estudio y eliminamos el *header* y el *index* para que los datos ya no presenten la primera columna indicando los nombres de los parámetros, ni los diferentes índices. Después, el parámetro "enfermedad" se transforma en la Y. Además, para realizar la regresión logística hemos definido las funciones estudiadas durante la asignatura. Estas son:

- **Función sigmoide:**

$$g(z) = \frac{1}{1 + e^{-z}}$$

```
# Sigmoide
def g(z):
    return 1 / (1 + np.exp(-z))
```

- **Función de coste:**

$$J(\theta) = -\frac{1}{m}((\ log\,(g(X\theta)))^T y + (\ log\,(1 - g(X\theta)))^T (1 - y))$$

```
def coste(Theta, X, y):
    m = np.shape(X)[0]
    Aux = (np.log(g(X @ Theta))).T @ y
    Aux += (np.log(1 - g(X @ Theta))).T @ (1 - y)
    return -Aux / m
```

- **Función gradiente:**

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{m} X^T (g(X\theta) - y)$$

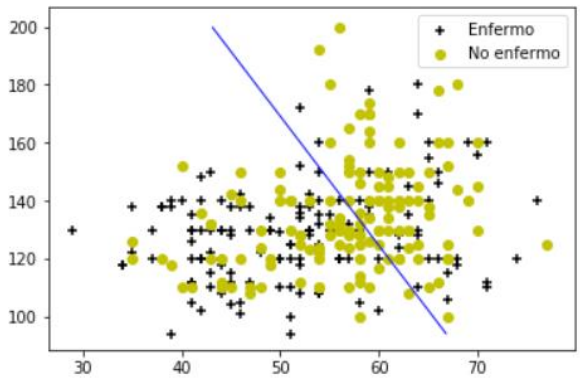
```
def gradiente(Theta, X, y):
    m = np.shape(X)[0]
    Aux = X.T @ (g(X @ Theta) - y)
    return Aux / m
```

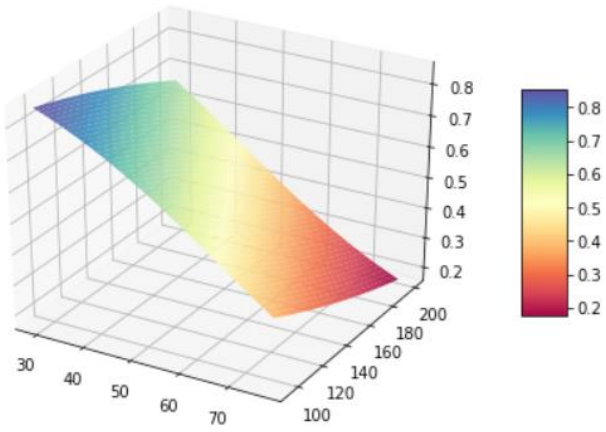
Gracias a estas funciones podemos hacer uso de la función *scipy.optimize.fmin_tnc* de SciPy para obtener los parámetros Theta que minimizan la función de coste para la regresión logística. Hemos implementado funciones para, una vez obtenidos los resultados, poder dibujar la frontera de decisión en una gráfica y saber cómo de preciso ha sido el experimento.

Además de esta regresión logística que utiliza tan solo dos parámetros elegidos, también hemos implementado otra que utiliza todos los parámetros al mismo tiempo. De esta manera la clasificación es más precisa.

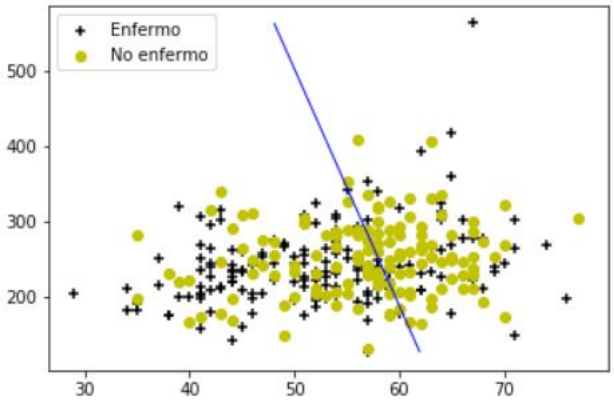
Resultados y Conclusiones

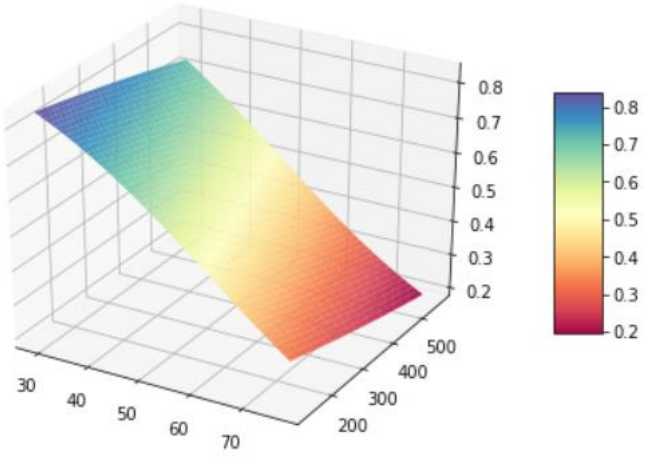
Primer experimento. Respecto a los parámetros de la edad y la tensión arterial. Presenta un porcentaje de acierto del 87%.





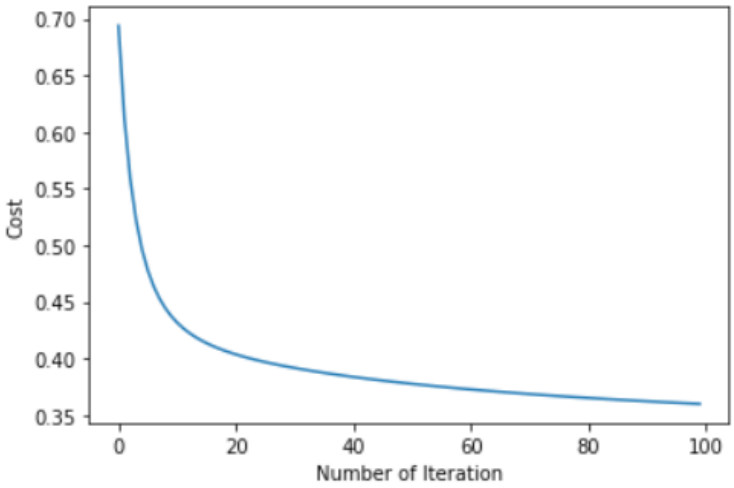
Segundo experimento. Respecto a los parámetros de la edad y el colesterol. Presenta un porcentaje de acierto del 83%.



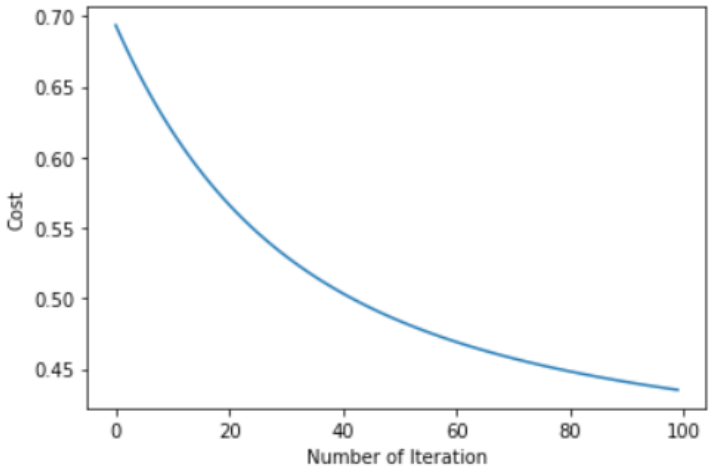


Tercer experimento. Regresión logística con todos los parámetros. Hemos hecho una comparación entre los resultados utilizando diferentes valores para Lambda.

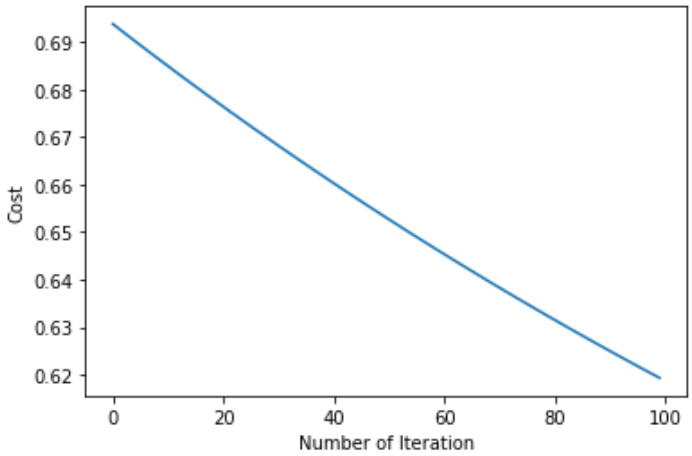
Con Lambda = 1.0, dando un coste aproximado de 0.359



Con Lambda = 0.1, dando un coste aproximado de 0.435



Con Lambda = 0.01, dando un coste aproximado de 0.6193



Se observa claramente la diferencia a la hora de seleccionar diferentes valores para Lambda, teniendo los valores más pequeños una curva menos definida y costes más altos.

Redes Neuronales

Las **redes neuronales** artificiales son un modelo computacional vagamente inspirado en el comportamiento observado en su homólogo biológico. Consiste en un conjunto de unidades, llamadas **neuronas artificiales**, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.

Cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un **valor de peso**. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Esta función se conoce como **función de activación**.

Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Para realizar este **aprendizaje automático**, normalmente, se intenta minimizar una función de pérdida que evalúa la red en su total. Los valores de los pesos de las neuronas se van actualizando buscando reducir el valor de la función de pérdida. Este proceso se realiza mediante la **propagación hacia atrás**.

Implementación

Para este proyecto vamos a reutilizar la arquitectura de red neuronal que utilizamos en las prácticas de la asignatura. En este caso, el **número de entradas** es mucho menor (21), al igual que el **número de salidas** (2), por lo que, necesariamente, la **capa intermedia oculta** contará también con un número menor de unidades neuronales (7). Esta **topología de tres capas** es invariable, ya que cambiarla supondría variar toda la arquitectura de la red neuronal y, por tanto, el código de la propagación hacia atrás. No es un modelo escalable, pero sirve para los propósitos de este proyecto.

La función de activación que utilizaremos, tanto en las capas de entrada y oculta como en la capa de salida, es la función sigmoidea, que ya hemos visto en el apartado de regresión logística. En esta ocasión, con motivo de la propagación hacia atrás, hemos tenido que implementar también la derivada de la función sigmoidea:

```
def sigmoid_derivative(Z):
    return sigmoid(Z) * (1 - sigmoid(Z))
```

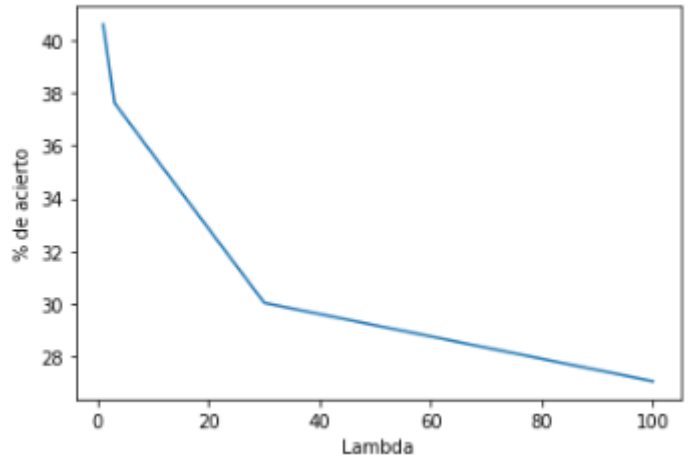
Las matrices de pesos que utilizará nuestra red neuronal para entrenarse se inicializan con valores aleatorios. Una vez construida la red neuronal, comenzamos a hacer pruebas con distintos valores para la tasa de aprendizaje Lambda y el número de iteraciones durante las que se lleva a cabo el proceso de optimización y aprendizaje.

Resultados y Conclusiones

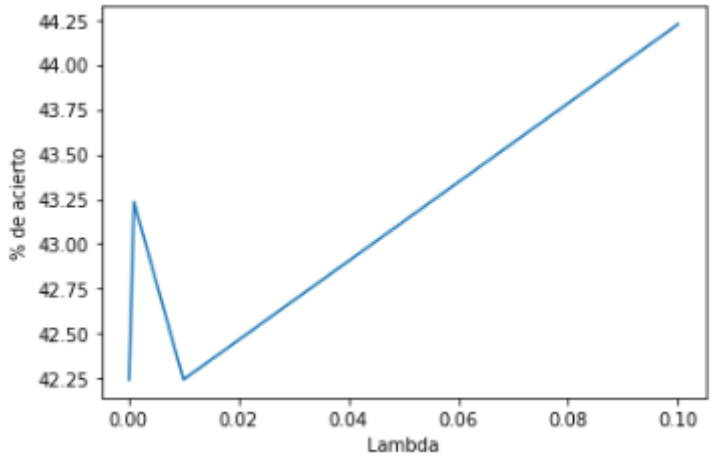
En un **primer experimento** con una tasa de aprendizaje muy baja (0’0001) y 750 iteraciones, nuestra red neuronal consigue una porcentaje de aciertos ciertamente cuestionable, tan solo un 42’57%. Probamos con otros valores tanto para la tasa de aprendizaje como para el número de iteraciones, pero los resultados no mejoran en ningún caso.

De este modo, nos planteamos realizar varias mediciones centradas en el valor de la tasa de aprendizaje, dejando el número de iteraciones fijo en 750; si intentamos más iteraciones el algoritmo suele fracasar, y con menos iteraciones los resultados son aún más bajos.

El primer grupo de tasas de aprendizaje lo componen cinco valores: 1, 3, 10, 30 y 100. Teóricamente, cuanto más se extremo sea el valor de Lambda peor será el desempeño de la red neuronal. Como podemos comprobar en la gráfica que obtenemos como resultado, el mejor porcentaje de acierto lo obtiene la tasa de aprendizaje 1.0:



Con estos datos, decidimos lanzar un **segundo experimento**, esta vez con tasas de aprendizaje inferiores a 1, para comprobar si el porcentaje de acierto de la red neuronal mejora. En esta ocasión seleccionamos los valores 0.0001, 0.001, 0.01 y 0.1, y los resultados son extraños. De forma general, la gráfica obtenida es la que se muestra a continuación:



Sin embargo, en un porcentaje elevado de repeticiones del experimento, nos encontramos con gráficas sin sentido o que presentan errores similares a los que aparecían cuando se elevaba en exceso el número de iteraciones. Estos fallos se producen con más frecuencia en los valores más pequeños de la tasa de aprendizaje (0.0001 y 0.001).

En cualquier caso, parece evidente que estos valores por debajo de 1.0 se comportan mejor que sus hermanos mayores, obteniendo porcentajes de acierto no inferiores al 42% y llegando hasta el 44’22%. El efecto de sesgo es mayor que el de *overfitting*, en este caso.

Support Vector Machines

Las máquinas de soporte vectorial (o **Support Vector Machines**, SVMs) son un conjunto de algoritmos de aprendizaje supervisado. Son métodos relacionados con problemas de **clasificación** y **regresión**. Con un conjunto de entrenamiento de muestras podemos entrenar una SVM para que prediga la clase de la siguiente muestra.

Por ello, las SVMs resultan muy útiles a la hora de estudiar nuestro dataset sobre las enfermedades cardiacas, ya que nos permite, dadas estas muestras, crear un sistema capaz de predecir si un paciente padece una enfermedad o no.

Implementación

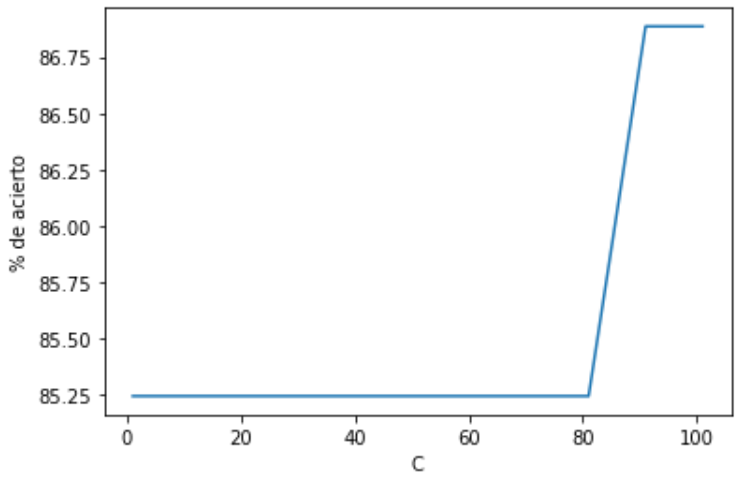
Para implementar las SVM, hemos utilizado *sklearn.svm* tal y como hemos hecho en la asignatura. Hemos definido la siguiente función:

```
def SVM_Linear(x_train, y_train, x_test, y_test, C):
    svm = SVC(C=C, kernel='linear', tol=1e-3, max_iter=-1)
    svm.fit(x_train, y_train)
    acc = svm.score(x_test, y_test) * 100
    return acc;
```

Resultados y Conclusiones

Hemos podido obtener diferentes resultados mediante la variación de C.

- Con C = 1 conseguimos un porcentaje de acierto del 85.25%.
- Con C = 10 conseguimos un porcentaje de acierto del 83.61%.
- Con C = 100 conseguimos un porcentaje de acierto del 86.89%.



El valor de C no parece afectar al porcentaje de acierto del algoritmo SVM hasta alcanzar valores cercanos a 80; a partir de ahí se produce un pequeño salto de dos puntos porcentuales, que se estabiliza al llegar a valores entorno a 90. En pruebas posteriores, con valores por encima de 100, el porcentaje de acierto no se ve alterado.

Comparativa con modelos de SKLearn

Una vez testados todos los modelos que implementamos a partir del código de las prácticas, pensamos que sería buena idea comparar los resultados con los de los diferentes módulos de la librería **SKLearn**. En esta ocasión nos centramos en comparar única y exclusivamente los porcentajes de acierto, dado que son la métrica más directa y, en cierto modo, la que más nos interesa.

El tratamiento previo de los datos es muy similar al utilizado para nuestras implementaciones, pero en esta ocasión simplificamos el paso de normalización utilizando la función **StandardScaler()** de SKLearn. Para facilitar la visualización de los resultados finales, utilizamos un diccionario que almacena el nombre del algoritmo empleado y el porcentaje de acierto obtenido tras el entrenamiento.

Regresión Logística

La función de SKLearn que nos permite aplicar regresión logística a nuestro dataset es **LogisticRegression()**. El porcentaje de acierto obtenido es de un 85'25% con los valores predeterminados de 100 iteraciones y tasa de aprendizaje de 1, algo inferior al 86'89% obtenido por nuestra implementación del algoritmo. Sin embargo, es de notar que no se aprecian anomalías con valores elevados de la tasa de aprendizaje (>10) que sí aparecían en nuestra implementación. Además, el porcentaje de aciertos de la función de SKLearn no presenta tanta variabilidad frente a los cambios.

```
lr = LogisticRegression()
lr.fit(x_train, y_train)
acc = lr.score(x_test, y_test) * 100
accuracies['Regresión Logística'] = acc
print("Test Accuracy {:.2f}%".format(acc))
```

Redes Neuronales

Como la instalación y manejo de **Keras** se complicó en exceso, decidimos que era mejor mantener este apartado de experimentos lo más uniforme posible y utilizar también uno de los módulos de alto nivel de SKLearn; de todos modos, SKLearn utiliza Keras (y, por tanto, TensorFlow) para sus módulos de redes neuronales.

Utilizamos la función **MLPClassifier()**, con un valor para la tasa de aprendizaje de 0.0001 y un número de iteraciones de 10000; por suerte, en esta ocasión el algoritmo no se rompe a mitad de ejecución, a pesar de estos valores tan extremos. La función de activación utilizada en las capas de entrada y en la capa oculta pasa a ser la **función relu**, pero en la capa de salida se mantiene la función sigmoidea para obtener los valores deseados entre 1 y 0. Configuramos, además, el tamaño de la capa oculta para que albergue 7 unidades neuronales, y el resultado que obtenemos es un porcentaje de acierto superior al 77%, significativamente mejor que el de nuestra implementación.

En cualquier caso, no es un resultado óptimo, y tras hacer varias pruebas con otras configuraciones de la red, e incluso con otros clasificadores y arquitecturas el porcentaje de aciertos no mejora sustancialmente, por lo que hemos concluido que las redes neuronales no son la mejor opción para clasificar los elementos de nuestro dataset y hacer predicciones.

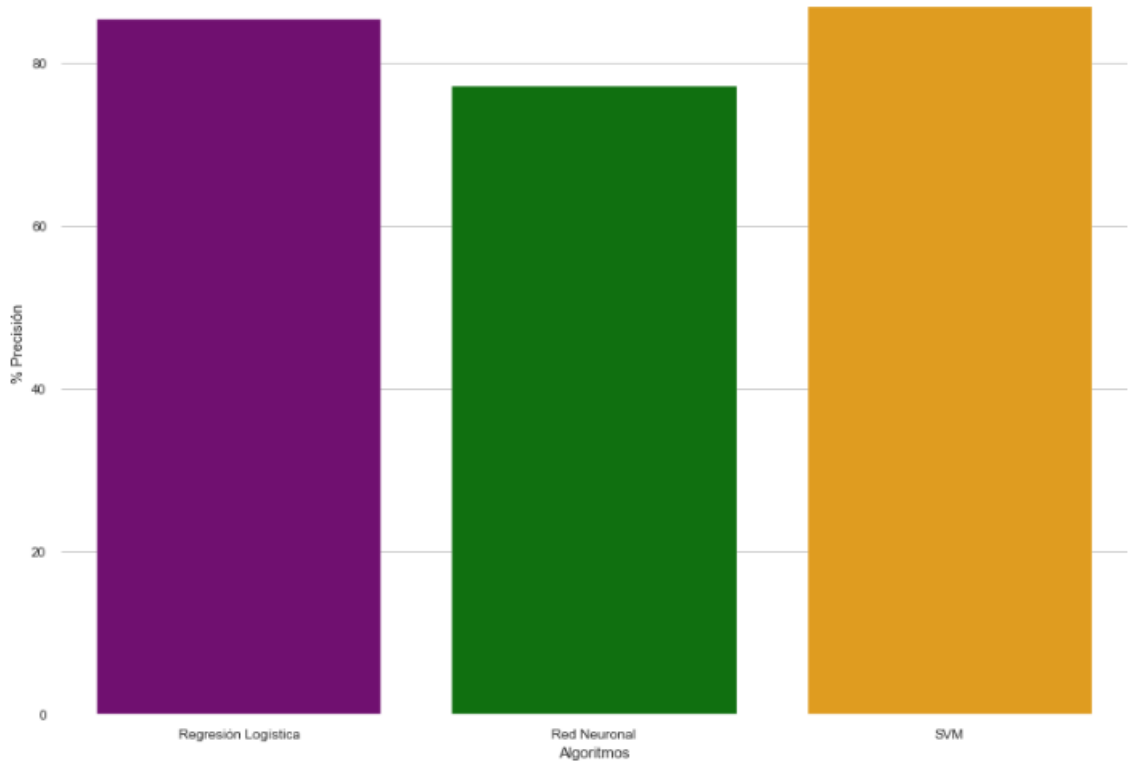
```
mlp = MLPClassifier(activation='relu', solver='sgd', alpha=1e-5, learning_rate_init=0.0001,
                    hidden_layer_sizes=(21, 7), max_iter=10000, random_state=200)
mlp.fit(x_train, y_train)
acc = (mlp.score(x_test, y_test)) * 100
accuracies['Red Neuronal'] = acc
print("Test Accuracy of NN: {:.2f}%".format(acc))
```

SVM

La función que utilizamos para SVM es la misma que aparecía en nuestra implementación, **SVC()**, pero su utilización es todavía más directa. El porcentaje de acierto obtenido es el más alto que hemos registrado con los modelos de SKLearn, con un 86'89%, a la par con nuestra implementación de la regresión logística y con la de las propias SVM con valores de C superiores a 90.

```
svm = SVC(random_state = 1)
svm.fit(x_train, y_train)
acc = svm.score(x_test, y_test) * 100
accuracies['SVM'] = acc
print("Test Accuracy of SVM: {:.2f}%".format(acc))
```

En esta gráfica podemos comparar la precisión de los tres algoritmos implementados con las herramientas que proporciona SKLearn:



Anexos

En el mismo directorio que este archivo se encuentra la carpeta *Code* que contiene todo el código utilizado para el desarrollo del proyecto. La mayor parte del mismo se ha llevado a cabo en **Jupyter Notebook**, por lo que los archivos incluidos están en formato HTML, de modo que sean completamente interactivos. Además, se incluye el archivo HTML generado por la librería Pandas Profiling, con la descripción pormenorizada del dataset.