UNIVERSIDAD DE GRANADA

Inteligencia de Negocio

grado en ingeniería informática

Práctica 1

Resolución de problemas de clasificación y análisis experimental.

Autor
José María Sánchez Guerrero

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2020-2021

Índice

1. Introducción

En este trabajo vamos a analizar el comportamiento de distintos algoritmos de clasificación en el problema propuesto. Disponemos de un dataset, llamado *"Mammographic Mass dataset"*, en el cual se desea predecir el tipo de tumor (benigno o maligno) en una serie de mamografías realizadas para un estudio sobre el cáncer de mama. Este estudio lo vamos a realizar gracias a los siguientes atributos proporcionados en el dataset:

- **BI-RADS.** Este parámetro representa un control de calidad de las mamografías. Consta de 7 categorías distintas, en las que, cuanto más alto sea el valor, hay una mayor probabilidad de que sea maligno.
- **Edad** del paciente.
- **Forma de la masa.** Dependiendo de como sea la masa anormal detectada, se clasifica como **R**edondeada, **O**valada, **L**obulada, **I**rregular ó **N**o definida.
- **Margen de masa.** Circumscribed = 1, microlobulated = 2, obscured = 3, ill-defined = 4, spiculated = 5 (nominal).
- **Densidad de la masa.** Valores entre 1 y 4, siendo 1 la más alta y 4 contenido graso (no tumoral).
- **Severidad.** Es el atributo que se desea predecir, es decir, si es un tumor benigno o maligno.

En el dataset hay datos de 961 pacientes, sin embargo, nos gustaría dejar un porcentaje para validar el modelo y así ver cómo va entrenando los datos. Posteriormente, se explicará cómo se ha determinado qué datos son los de entrenamiento y cuáles son los de test.

2. Procesado de datos

Lo primero que tenemos que hacer es mostrar varios de los datos que tenemos y analizarlos. En mi caso vamos a sacar las 5 primeras filas:

	BI-RADS	Age	Shape	Margin	Density	Severity
0	5.0	67.0	L	5.0	3.0	maligno
1	4.0	43.0	R	1.0	NaN	maligno
2	5.0	58.0	I	5.0	3.0	maligno
3	4.0	28.0	R	1.0	3.0	benigno
4	5.0	74.0	R	5.0	NaN	maligno

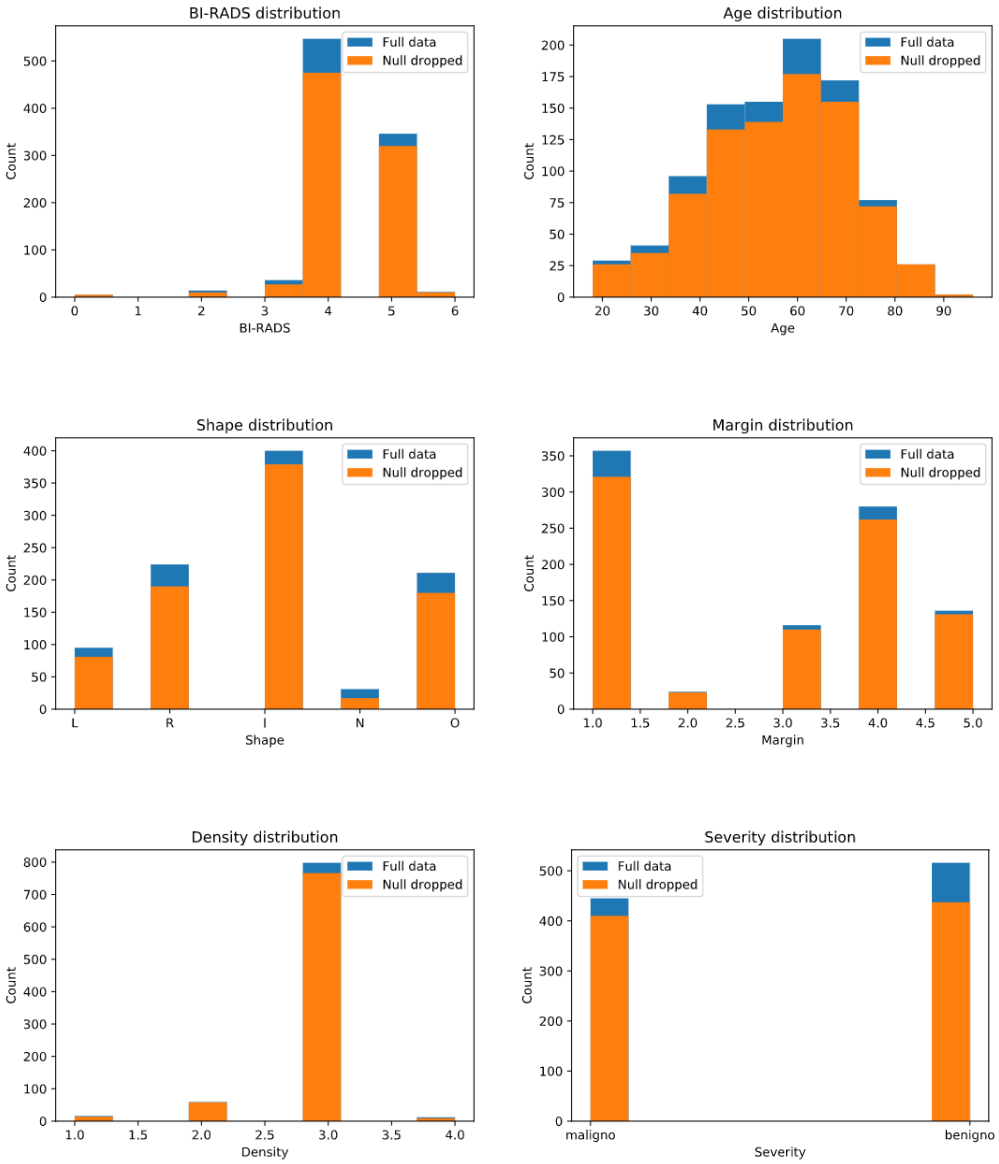
Podemos observar que tenemos tanto datos numéricos, como datos categóricos, como ya comentamos en la introducción. También podemos observar que tenemos varias celdas con datos erróneos o perdidos (representados con el valor NaN), por lo que será importante procesarlos para que nuestros algoritmos funcionen correctamente. Primero veamos qué cantidad de estos datos nulos tenemos:

BI-RADS	2
Age	5
Shape	0
Margin	48
Density	76
Severity	0

Son una cantidad bastante alta de datos, en comparación con la cantidad de datos totales que tenemos. Por lo tanto, eliminar toda las filas que contengan uno, puede dejarnos con muy pocos datos para entrenar y validar, y que el modelo sea más débil. No obstante, el introducir datos para reemplazar uno faltante ha de realizarse con cuidado, ya que no son datos reales.

También tenemos que tener en cuenta cuál es la distribución de estos datos antes de trabajar con ellos. Es decir, tenemos que asegurarnos de no sesgar nuestros datos si los eliminamos. Si hay algún tipo de correlación, tendríamos que intentar completarlos de alguna forma. Para ello, vamos a generar una gráfica para cada uno de los atributos del *dataset*, en la que mostraremos la cantidad de datos antes y después de eliminarlos, y asi ver cómo están distribuidos.

Los resultados son los siguientes:



Podemos observar que los datos nulos están distribuidos aleatoriamente entre los atributos, por lo que podremos eliminarlos del *dataset* sin ningún problema (y teniendo en cuenta que tendremos menos datos para trabajar).

3. Configuración de algoritmos

Para todos los algoritmos hemos procedido de la misma manera, y así evaluarlos a todos en igualdad de condiciones. Para comenzar, declaramos el clasificador con sus parámetros correspondientes (en nuestro caso, las primeras evaluaciones han sido con los parámetros por defecto y una semilla $random\_state = 0$).

Posteriormente, hacemos las predicciones correspondientes para los datos de entrenamiento. Lo hacemos mediante validación cruzada de 5 particiones, gracias a la función:

$$cross\_val\_predict(classifier, x\_train, y\_train, cv = 5)$$

Por último, para evaluar los resultados obtenidos vamos a usar:

- **Classification report.** Crea un informe que muestra las principales métricas de clasificación: precisión, recall, f1-score, y promedios macro, ponderado y de la muestra.
- **Score.** Misma medida del informe anterior mostrada con un poco más de precisión.
- **AUC Score.** Métrica que calcula el área bajo la curva ROC generada a partir de las predicciones.
- **Confusion matrix.** Muestra una matriz para evaluar la precisión de la clasificación. Cada fila representa las instancias de una clase predicha, mientras que cada columna representa las instancias reales de ésta.

Los algoritmos que evaluaremos han sido elegidos porque, cada uno de ellos, tiene una forma de procesar los datos diferente a todos los demás. Estos algoritmos son los siguientes:

3.1. K-Nearest-Neighbors (k-NN)

Comenzamos por este algoritmo ya que es uno de los más utilizados, debido a su simplicidad. Este algoritmo funciona de la siguiente manera. Cuando tenemos un nuevo ejemplo a clasificar, calcula la distancia (Euclídea) con respecto a los datos ya existentes, y considerando los k más cercanos, determina si pertenece a una clase u otra.

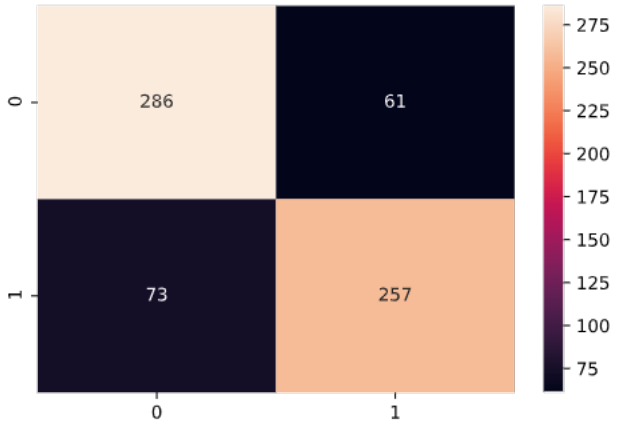
A la hora de implementarlo lo hacemos de la siguiente forma:

$$KnnClf = KNeighborsClassifier(n\_neighbors = 5)$$

Seleccionamos un número **k=5** de vecinos a evaluar y dejamos el resto de parámetros que trae el algoritmo por defecto. El parámetro anterior es el más relevante, sin embargo, algunos otros interesantes a estudiar pueden ser, por ejemplo: **p**, que sirve para cambiar el tipo de distancia utilizada (Euclídea, Manhattan o Minkowski); ó **weights**, que determina la influencia de los vecinos en la predicción (todos ponderan igual, los cercanos influyen más o tu propia función).

Resultados obtenidos

	precision	**recall**	**f1-score**	**support**
0	0.80	0.82	0.81	347
1	0.81	0.78	0.79	330
accuracy			0.80	677
macro avg	0.80	0.80	0.80	677
weighted avg	0.80	0.80	0.80	677



SCORE: 0.7991137370753324

AUC score: 0.7986900707361801

3.2. Decision Tree

Este clasificador, como su propio nombre indica, es un árbol en el que cada **hoja** es una clase y cada **nodo** es un nodo de decisión con una prueba simple a realizar.

El algoritmo funciona de la siguiente forma. Todos los ejemplos de entrenamiento comienzan desde el nodo raíz y se dividen recursivamente los ejemplos en base a los atributos seleccionados. Esta selección de atributos se suele realizar mediante el criterio "*gini*", pero también hay otros como "*InfoGain*" o "*GainRatio*". Posteriormente, quitamos las ramas con ruido o con datos anómalos.

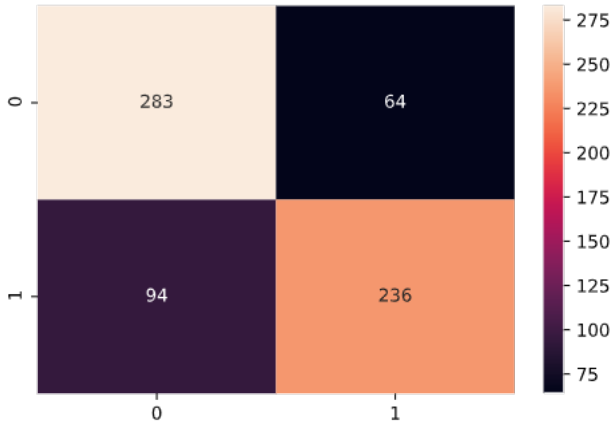
A la hora de implementarlo lo hacemos de la siguiente forma:

$$DecTreeClf = tree.DecisionTreeClassifier(random_state = 0)$$

El parámetro **random_state** nos sirve para seleccionar la semilla. El resto de parámetros los dejamos por defecto, ya que la mayoría son utilizados para limitar el número de hojas, el número de características, etc. que en nuestro ejemplo no necesitaremos. También tenemos el parámetro **criterion**, que nos sirve para elegir el criterio de selección de atributos comentado anteriormente.

Resultados obtenidos

	precision	**recall**	**f1-score**	**support**
0	0.75	0.82	0.78	347
1	0.79	0.72	0.75	330
accuracy	0.77	677	0.80	677
macro avg	0.77	0.77	0.77	677
weighted avg	0.77	0.77	0.77	677



SCORE: 0.7666174298375185

AUC score: 0.7653567374028469

3.3. Naive-Bayes

Se ha utilizado porque es el modelo de red bayesiana orientada a clasificación más simple. Este algoritmo se considera un estándar y sus resultados son competitivos, pese a utilizar una hipótesis poco realista. Es decir, suponemos que todos los atributos son independientes a partir de su clase, asi que la hipótesis MAP (Máximo a posteriori) en un Naive-Bayes queda así:

$$C_{MAP} = \arg_{c\epsilon\Omega_C} maxP(c|a_1, ..., a_n) = \arg_{c\epsilon\Omega_C} maxP(c) \prod_{i=1}^{n} P(a_i|c)$$

En la práctica, existen dependencias entre variables, por lo que puede llevar a una falta de precisión que no se puede retocar en un clasificador como este (existen redes de creencia bayesianas para solucionar esto).

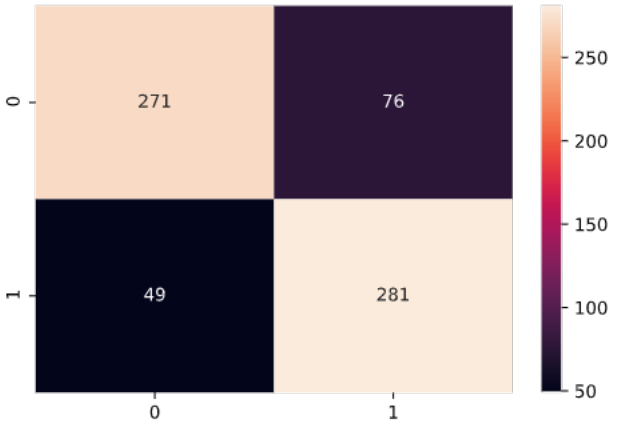
A la hora de implementarlo lo hacemos de la siguiente forma:

$$GaussianClf = GaussianNB()$$

No ponemos ningún parámetro ya que es un clasificador que no se puede modelar o retocar, como acabamos de decir.

Resultados obtenidos

	precision	recall	f1-score	support
0	0.85	0.78	0.81	347
1	0.79	0.85	0.82	330
accuracy	0.82	677	0.80	677
macro avg	0.82	0.82	0.82	677
weighted avg	0.82	0.82	0.82	677

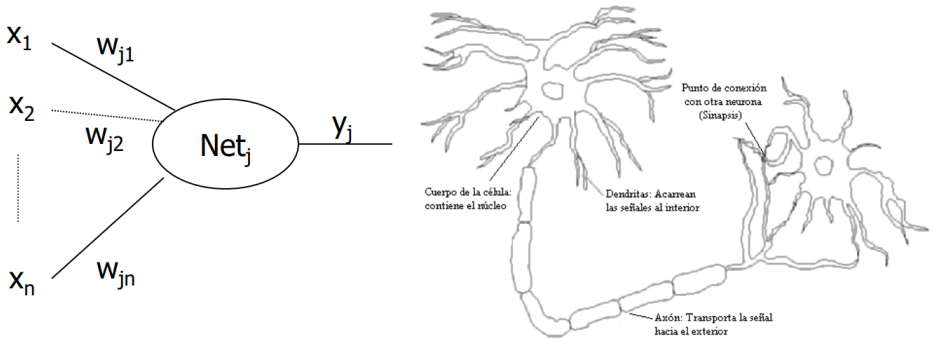


SCORE: 0.8153618906942393

AUC score: 0.8162474893022443

3.4. Neural Network

Estos algoritmos están basados en los propios sistemas nerviosos biológicos:



Como podemos ver en la figura, cada señal que llega a las dendritas $\{x_1, x_2, ..., x_n\}$ serían nuestros datos de entrada del modelo. Las sinapsis o puntos de conexión con otra neurona, serían los pesos $\{w_1, w_2, ..., w_n\}$ que ponderan a las entradas: **positivo** es una sinapsis excitadora y **negativo** es una sinapsis inhibidora.

Después, la actividad interna de cada célula sería la sumatoria entre cada una de las conexiones anteriores, y por último, la **función de activación** que, dadas estas entradas, define una salida para el modelo:

$$y_j = f(Net_j - \theta_j) = f(\sum w_{ji} \cdot x_i - \theta)$$

Para este trabajo, no vamos a implementar una red neuronal desde cero, ya que es una tarea bastante compleja. Sin embargo, vamos a utilizar un **MultiLayer Perceptron (MLP)**, que es una red neuronal compuesta por: una capa de entrada, capas ocultas (o *hidden layers*) y una capa de salida. Excepto los datos de entrada, cada nodo es una neurona que utiliza una función de activación no lineal (sigmoidal es la más común), y en el entrenamiento utiliza la técnica de *backtracking*

A la hora de implementarlo lo haremos de la siguiente forma:

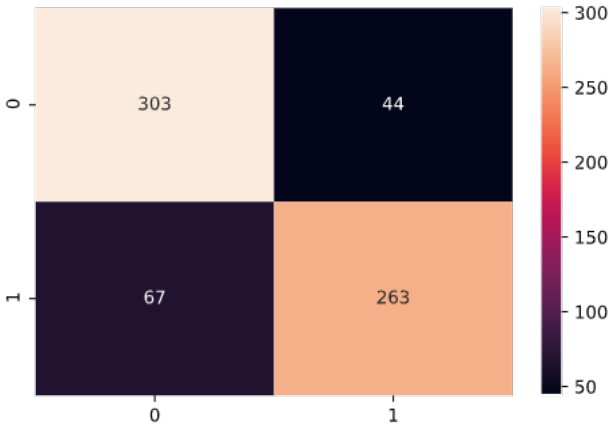
$$PerceptronClf = MLPClassifier(hidden\_layer\_sizes = 100, random\_state = 0)$$

Aquí podemos ver la cantidad de capas ocultas que queremos que tenga nuestra red. Es el parámetro más importante, ya que cuanto más alto sea, más aprenderá de los datos que tenemos. Esto no quiere decir que tengamos que poner un valor muy alto, ya que puede causar sobreaprendizaje en nuestro modelo. El otro parámetro que aparece, al igual que en los otros clasificadores, nos sirve para seleccionar la semilla.

También existen otros parámetros importantes, como pueden ser: **activation**, con la que podemos seleccionar la función de activación a utilizar; **early_stopping**, que sirve para finalizar el entrenamiento si no se mejora durante un número X de iteraciones; o parámetros relacionados con los pesos como **solver** (para la optimización de los pesos), **learning_rate** (tasa de aprendizaje) ó **tol** (tolerancia).

Resultados obtenidos

	precision	**recall**	**f1-score**	**support**
0	0.82	0.87	0.85	347
1	0.86	0.80	0.83	330
accuracy	0.84	677	0.80	677
macro avg	0.84	0.84	0.84	677
weighted avg	0.84	0.84	0.84	677



SCORE: 0.8360413589364845

AUC score: 0.8350842721159724

3.5. Support Vector Machine (SVM)

El último algoritmo que vamos a utilizar es uno de los más populares en cuanto a la clasificación binaria. Una SVM construye un hiperplano entre las dos clases de forma que la separación entre ellas se amplie al máximo. Se van transformando los datos de entrada en un espacio de caracteristicas mediante el **producto interno (escalar)**.

A la hora de implementarlo lo haremos de la siguiente forma:

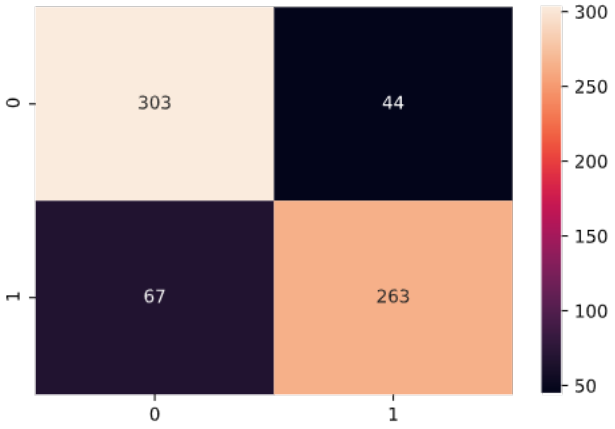
$$SvmClf = SVC(random_state = 0, gamma =' auto')$$

El primer parámetro que tenemos aquí es la semilla, al igual que en los otros. El siguiente parámetro es para seleccionar el coeficiente del **kernel** (el cual también es otro parámetro que también se podrá modificar).

También tenemos otros parámetros como son la **C**, para la regularazación; **tol** o tolerancia, como en las redes neuronales; o hasta un **max_iter** por si queremos poner un máximo de iteraciones.

Resultados obtenidos

	precision	recall	f1-score	support
0	0.81	0.88	0.84	347
1	0.86	0.78	0.82	330
accuracy	0.83	677	0.80	677
macro avg	0.83	0.83	0.83	677
weighted avg	0.83	0.83	0.83	677



SCORE: 0.8301329394387001

AUC score: 0.8289494367304165

4. Análisis de los resultados

Podemos observar que tenemos aproximadamente un 80 % de precisión, es decir, el modelo ha acertado un 80 % de los casos que se le ha propuesto. No

5. Interpretación de los resultados

6. Conclusión

7. Bibliografía