Práctica 1: Resolución de problemas de clasificación y análisis experimental

Alejandro Alonso Membrilla

Grupo: viernes
Correo: aalonso99@correo.ugr.es

Contents

1 Introducción **3**

2 Procesado de los datos **3**

3 Configuración de algoritmos **7**

4 Resultados obtenidos **9**

5 Análisis de resultados **18**

6 Interpretación de los datos **20**

7 Contenido adicional **21**

8 Bibliografía **23**

1 Introducción

La clasificación de conjuntos de datos es un problema ampliamente estudiado y de gran utilidad práctica en casi cualquier rama de la ciencia y de la ingeniería. El objetivo de esta práctica es la prueba, estudio y comparación de diversos modelos de aprendizaje supervisado, una serie de métodos que permiten aplicar aprendizaje estadístico sobre un dataset etiquetado para poder clasificar otros casos similares con etiquetas desconocidas.

En general, el problema de clasificación puede formularse matemáticamente como sigue:

Sean $X = \{x_1, ..., x_n\} \subset P = P_1 \times ... \times P_r$ *un conjunto de n vectores r-dimensionales donde r es el número de parámetros observados para cada instancia del conjunto, y* $P_i, 1 \leq i \leq r$, *es el conjunto de posibles valores del parámetro i-ésimo de los elementos de X. Sea* $y = \{y_1, ..., y_n\}$ *un vector de* C^n, *donde C es el conjunto de posibles tipos en los que se pueden clasificar los elementos de X. y representa, por tanto, las etiquetas de las instancias del conjunto X. El problema de clasificación consiste en encontrar una función* $f : P \to C$ *que mantenga la etiqueta dada a cada* $x_n \in X$.

El problema de clasificación se diferencia del problema de regresión en que C contiene valores discretos y, en la práctica, finitos. Una solución al problema puede evaluarse tomando otro subconjunto X' de P y un conjunto de etiquetas y' para sus elementos, y comprobando la similitud entre $(y'_1, ..., y'_n)$ y $(f(x'_1), ..., f(x'_n))$.

El caso que nos ocupa es el de la clasificación de tumores detectados en mamografías como "malignos" o "benignos", para el posterior diagnóstico del paciente. Son de interés dos posibles enfoques del aprendizaje: el de la interpretabilidad del modelo, y el de la minimización del error.

- **Interpretabilidad:** en el ámbito del estudio de una enfermedad nueva o con comportamientos desconocidos, puede ser de gran interés para la comunidad científica encontrar, por medio de la computación, métodos directos para clasificar la enfermedad de un paciente en base a sus síntomas. Con este objetivo, un modelo interpretable da mucha más información útil para posterior investigación.
- **Precisión y fiabilidad:** en caso de que los resultados de aquellos modelos interpretables no sean suficientemente buenos o no arrojen nueva información sobre el fenómeno estudiado, tiene sentido dar prioridad a aquellos métodos que resuelvan el problema de clasificación con un menor margen de error.

En el desarrollo de esta práctica procuraremos aplicar una amplia variedad de modelos al problema de clasificación de tumores, utilizando el dataset proporcionado para entrenar y testar sus respectivos resultados, compararlos y estudiar en cierta medida un posible ajuste de parámetros de cada estimador. Finalmente, intentaremos dar la mejor solución posible a la clasificación tanto en cuestión de precisión como de comprensión de los datos.

El total del procesamiento de datos y los cálculos estadísticos realizados para este trabajo han sido implementados en `Python` usando las APIs `pandas` para el manejo de tablas de datos etiquetados, `matplotlib` y `seaborn` para el dibujado de gráficos, `numpy` para el cálculo vectorial y `scikit-learn` para el entrenamiento de los modelos y diversos procesamientos de datos realizados. La web de `sklearn` ha sido particularmente útil, puesto que en su guía de usuario explica el uso y conceptos generales de las técnicas que implementa, y ha sido un material de apoyo muy completo para esta práctica. El enlace a la página de cada biblioteca está disponible en las referencias.

2 Procesado de los datos

El procesado de los datos suministrados ha constado de diferentes fases.

Lectura de los datos

En primer lugar, tras la carga del fichero con los datos y una impresión y lectura rápida de los mismos, observamos varias características destacables:

- El dataset cuenta con 961 instancias, y 5 parámetros o variables distintas por instancia.

- Todos los datos de la tabla son enteros, pero están formateados como flotantes.
- Todas las columnas de la tabla, incluyendo obviamente la de la etiqueta, son categóricas.
- Hay valores perdidos. Las columnas más perjudicadas son la del margen de masa (48 pérdidas) y la de la densidad de la masa tumoral(76 pérdidas).

Valores perdidos

Antes de entrenar modelos de predicción usando nuestro conjunto de datos es importante eliminar/sustituir los valores perdidos para poder pasar a los algoritmos de aprendizaje valores numéricos con los que este pueda trabajar.

Si una columna está formada mayoritariamente por valores perdidos, esta se puede eliminar. La columna con más pérdidas en nuestro caso es "Density" con 76 pérdidas, menos de un 10%, lo que no es suficiente para eliminarla. Por otro lado, tampoco parece conveniente eliminar 76 filas completas de la tabla. Intentaremos, por tanto, sustituir estos valores de la forma más insesgada que encontremos.

Sustituir los valores perdidos por la moda de su respectiva columna es realizable en columnas como BI-RADS y Age con ínfimas pérdidas. Para el resto de columnas, puede suponer un sesgo que es mejor evitar.

En su lugar, probaremos con el IterativeImputer de Scikit-Learn. Este método utliza valores flotantes, por lo que redondearemos el resultado para obtener un resultado válido en aquellas columnas que solo acepten valores enteros (en nuestro caso, todas).

Búsqueda de outliers

Sin información adicional sobre la naturaleza de los datos que estamos manejando, **no podemos realizar un estudio general de los outliers** en este conjunto porque las variables son cualitativas.

Rasgos generales sobre las etiquetas

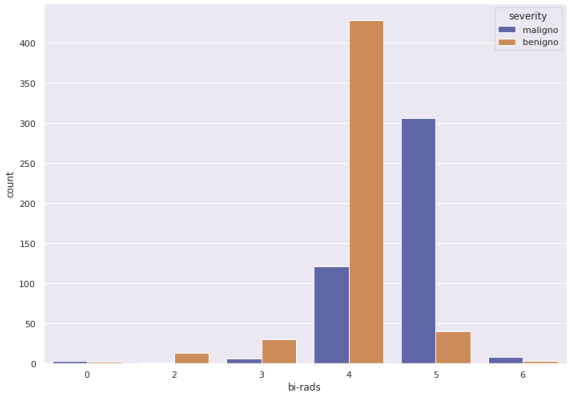
La clasificación de los tumores puede ser de 'benigno' o 'maligno'. Saber cuántos hay de cada uno nos permitirá intuir cuan buenos son nuestros modelos y nos será útil, en general, en la fase de análisis de resultados. En nuestro caso, vemos que hay 961 tumores en total, de los cuales 516 (un 53.69%) son benignos. Esto implicará que, si asumimos que todos los tumores son benignos acertaremos un 53.69% de las veces. Este umbral a superar es bastante bajo, por lo que no debería ser difícil obtener resultados mejores.

Dependencia

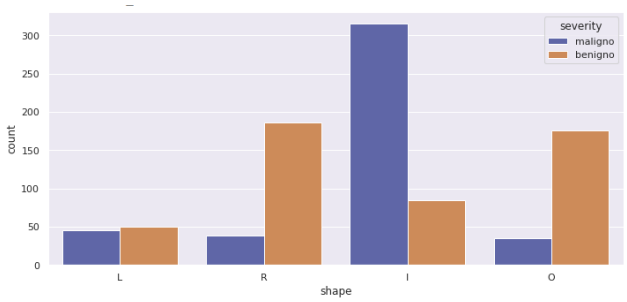
Para cada parámetro de nuestro dataset, vamos a visualizar un histograma que represente el número de tumores benignos o malignos dependiendo de cada valor de dicho parámetro. El objetivo es obtener una idea general de la dependencia del resultado de la clasificación con cada una de las variables explicativas. La figura 1 muestra los histogramas con los resultados.

Observamos los siguientes patrones:

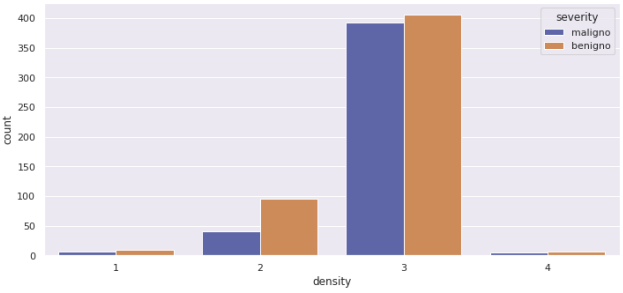
- Una categoría BI-RADS de 4 indica que el tumor probablemente será benigno, mientras que una de 5 casi asegura su malignidad.
- Una forma irregular presenta una alta probabilidad de pertenecer a un tumor maligno, mientras que si su forma es redonda u ovalada probablemente sea benigno.
- Un margen de masa de tipo 1 corresponderá probablemente a un tumor benigno mientras que si es de tipo 5, aunque de estos hay una menor cantidad, probablemente el tumor se descubra como maligno.



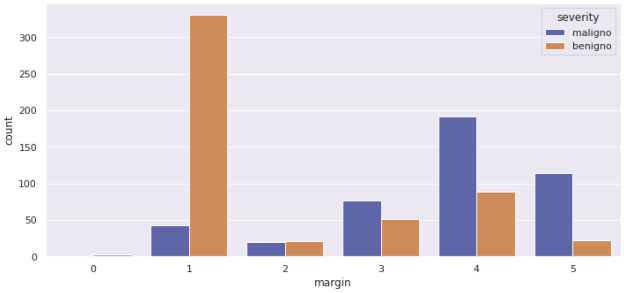
BI-RADS



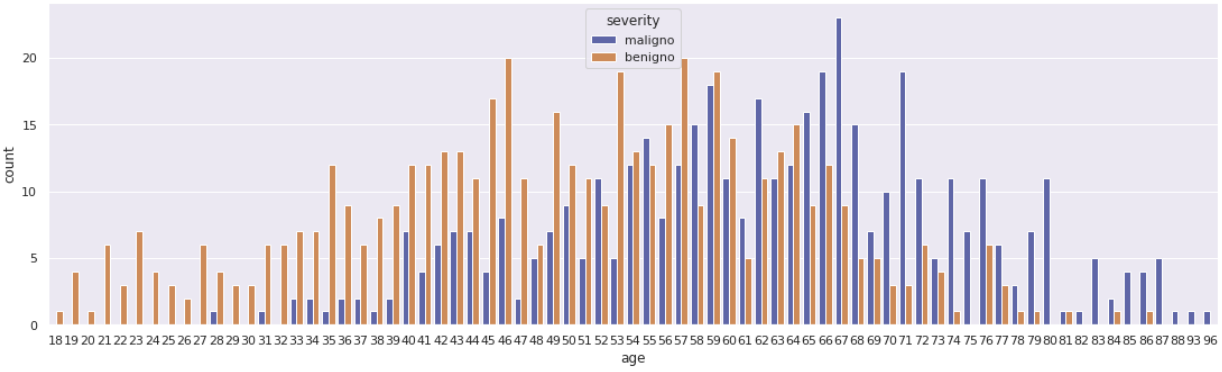
Forma



Densidad de la masa tumoral



Margen de masa



Edad

Figure 1: Estudio de las variables del dataset y la dependencia de los valores de la etiqueta para con las mismas.

- Las cantidades de tumores benignos y malignos se distribuyen sobre las distintas edades de forma similar a una normal, pero con medias desplazadas. La distribución de los tumores benignos se encuentra desplazada hacia edades más jóvenes, mientras que en las edades más avanzadas la probabilidad de que el tumor sea maligno aumenta.

Análisis de Componentes Principales

Nuestro conjunto de datos tiene 5 dimensiones (6 si contamos el valor de la etiqueta). No es, por tanto, representable en un gráfico 2D. Para paliar este problema de forma que nos sea posible visualizar el conjunto de datos hemos aplicado un Análisis de Componentes Principales.

Los resultados obtenidos son los siguientes. Los pesos (cantidad de información/varianza encontrada en ese eje respecto al conjunto total) son de:

$$(9.81290006e-01 \quad 1.38315225e-02 \quad 2.16787483e-03 \quad 1.83408888e-03 \quad 8.76507512e-04)$$

Una rápida cuenta nos confirma que las dos primeras componentes en el espacio de datos transformado contienen más del 99,9% de la información del dataset. Nuestro dataset proyectado sobre esos ejes (cuyo valor particular no se mostrará, al no ser relevante) luce de la siguiente forma:

Los puntos rojos representan los tumores clasificados como malignos, y los azules a los clasificados como benignos. Se aprecia una separación relativamente clara de ambas clases, pero también que existe una superposición entre ellas que será muy difícil de clasificar.

Selección de características

Tratamos de encontrar las mejores características aplicando tests estadísticos univariables. Usamos el módulo SelectKBest de Scikit-Learn con el test de χ^2 . La puntuación para cada columna es la siguiente:

$$(28.32051067 \quad 674.99567216 \quad 353.2295735 \quad 276.67370836 \quad 0.7646782)$$

El selector de características utilizado parece indicar que la propiedad Density cuenta con una puntuación como característica muy baja, lo que es razonable teniendo en cuenta que su valor es casi siempre el mismo. El código BI-RADS tampoco tiene una puntuación muy alta. Tendremos esto en cuenta en la fase de entrenamiento de los modelos.

Codificación OneHot

La mayor parte de las variables en este estudio no son cuantitativas sino, como ya se ha mencionado anteriormente, cualitivas. Esto hace que la segmentación espacial que llevan a cabo muchos de los algoritmos utilizados no tengan sentido, porque no existe ningún orden entre los posibles valores de cada categoría. Como codificación alternativa se ha probado a transformar el valor de cada variable en un array binario (one-hot array) con el módulo OneHotEncoder de sklearn. Los resultados obtenidos con esta codificación y el análisis de los mismos se verán en los apartados 4 y 5, respectivamente.

3 Configuración de algoritmos

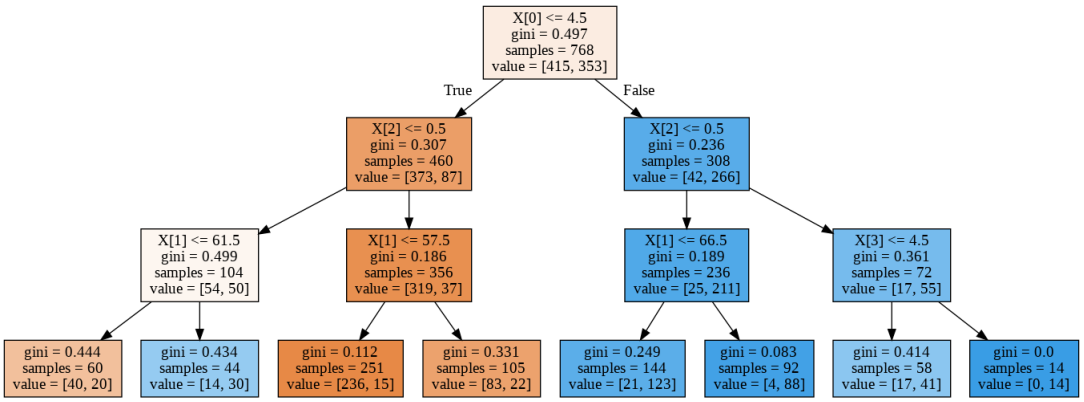
Decision Tree

Los árboles de decisión son los modelos más interpretables de los que se han utilizado en este estudio. Es por esto que encontrar una configuración que permita una interpretación sencilla del modelo es especialmente importante. En este caso se ha hecho una comparación de los resultados para diversas profundidades del árbol generado, que puede verse en la siguiente tabla:

	Precisión	AUC	F1-Score
Árbol completo	0.7659	0.763	0.7411
Árbol p1	0.8189	0.8111	0.783
Árbol p2	0.8127	0.8127	0.8009
Árbol p3	0.8356	0.8304	0.8106
Árbol p4	0.8179	0.8176	0.8053
Árbol p5	0.8189	0.8159	0.7986

El árbol completo genera nuevas ramas hasta haber clasificado la totalidad de los elementos del dataset de entrenamiento en hojas distintas. Podemos deducir, por tanto, que el hecho de que haya obtenido la peor puntuación se debe a que el número de instancias del conjunto de datos sea mucho mayor al número de parámetros, produciéndose *overfitting*. La profundidad que ha dado mejores resultados es 3, lo que encaja con la selección de variables que hemos realizado en el apartado anterior.

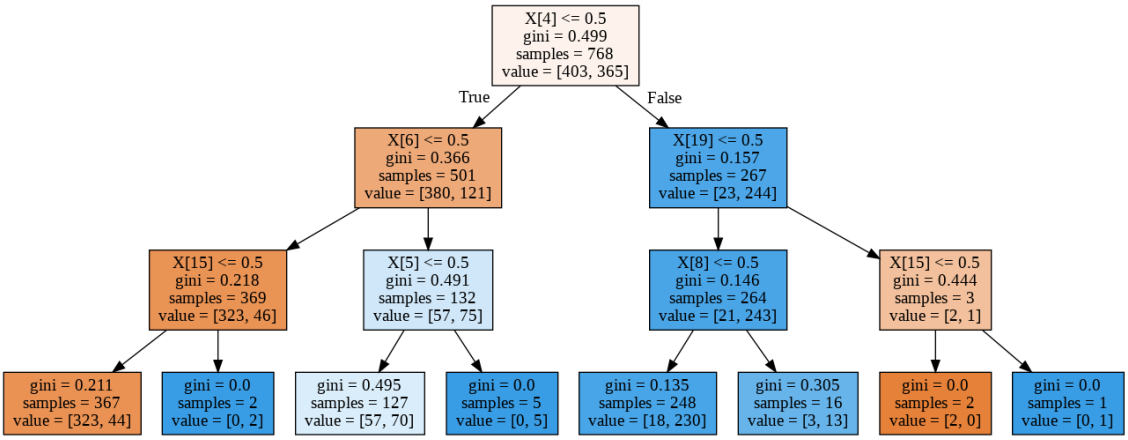
Fijando la profundidad a 3, a continuación vemos un ejemplo de árbol de decisión entrenado a partir de 4/5 de las instancias del dataset:



Nota: X[i] representa el valor en la i-ésima columna del elemento que se esté clasificando.

La clasificación hecha por el árbol es fácilmente interpretable.

Aprovechamos para explicar que la codificación de las variables categóricas en arrays OneHot podría hacer que la clasificación con árboles de decisión tenga más sentido. Sin embargo también hace que el modelo sea menos interpretable. A continuación se muestra el árbol de profundidad 3 para la versión codificada:



Ahora cada variable es un 0/1 en la codificación binaria de cada instancia y, por tanto, no las podemos interpretar.

AdaBoost

AdaBoost consiste en la creación de un clasificador complejo a partir de la unión de múltiples clasificadores sencillos, que se van optimizando iterativamente.

En el caso de esta algoritmo de boosting, los parámetros que pueden estudiarse son, principalmente, el número de estimadores y el ratio de aprendizaje. Si bien no se hemos realizado una experimentación exhaustiva con la modificación de dichos parámetros, hemos comparado dos casos:

- **Ratio de aprendizaje alto ($\alpha = 1$):** Funciona mejor para un bajo número de estimadores ($N_e = 5$). Los resultados son

  Precisión: 0.8252
  AUC: 0.8222
  F1-Scores: 0.8056

- **Ratio de aprendizaje bajo ($\alpha = 0.1$):** Funciona mejor para un bajo número de estimadores ($N_e = 30$). Los resultados son

  Precisión: 0.8377
  AUC: 0.8331
  F1-Scores: 0.8147

Notamos que aumentar el número de estimadores y reducir el ratio de aprendizaje (lo que aumenta el tiempo de cómputo pero favorece la robustez del algoritmo) ayuda a mejorar ligeramente los resultados en este caso.

Regresión Logística

Hemos probado el algoritmo con dos intensidades de regularización distintas. Si C es la inversa de dicha intensidad, obtenemos que:

- $C = 1$**:** los resultados son

  Precisión: 0.8189
  AUC: 0.8187
  F1-Scores: 0.8067

- $C = 0.1$**:** Mayor regularización. Los resultados son

Precisión: 0.8252
AUC: 0.8255
F1-Scores: 0.8146

Vemos que el estimador obtiene resultados ligeramente mejores al aumentar la regularización de los datos de entrenamiento.

Random Forest

De forma análoga al árbol de decisión, Random Forest funcionará mejor podando sus ramas más profundas. Al ser un multiclasificador este modelo pierde la interpretabilidad del árbol simple, pero esperamos obtener resultados mejores. Podemos verlos en la siguiente tabla:

	Precisión	AUC	F1-Score
Árboles completos	0.7919	0.7906	0.7748
Árboles p1	0.8252	0.8183	0.7936
Árboles p2	0.8252	0.8258	0.8154
Árboles p3	0.8293	0.8275	0.8132
Árboles p4	0.8398	0.8384	0.8258
Árboles p5	0.8335	0.832	0.8186
Árboles p6	0.8293	0.8276	0.8136
Árboles p7	0.8273	0.8259	0.8122

Una profundidad de 4 es la que obtiene mejores resultados en el Random Forest. Nos quedaremos con esta versión para los apartados posteriores.

Redes Neuronales

El estudio de los parámetros óptimos para el entrenamiento de modelos neuronales es un campo de investigación muy amplio y complejo. En esta práctica no se han comparado distintas redes ni sus resultados puesto que obtener soluciones óptimas requeriría mucho tiempo, pero merece la pena señalar que podrían utilizarse diversos algoritmos evolutivos, como los genéticos, capaces de resolver problemas de optimización en espacios de soluciones complejos.

4 Resultados obtenidos

En este apartado mostraremos el código de creación de cada modelo probado y los resultados obtenidos.

A modo de anotación, el dataset "normalizado" ha sido trasladado (restándole su valor mínimo) y escalado (dividiéndolo entre la diferencia entre su valor máximo y su valor mínimo) para normalizarlo al intervalo [0,1].

Código general

Las funciones siguientes en python se utilizan de forma recurrente en la creación de los modelos.

```
#reductor solo se usa si show_plot es True
def test_model(estimator, X, y, reductor=None, cv=5, show_plot=False):
    y_pred = cross_val_predict(estimator, X, y, cv=cv)
    if show_plot:
        assert len(X.shape) == 2 or reductor != None
        red_data = reductor.transform(X)[:,0:2]
```

```
            #Visualizamos los clasificados en cada clase
            x_1 = [d[1] for i,d in enumerate(red_data) if y_pred[i]==1.0]
            y_1 = [d[0] for i,d in enumerate(red_data) if y_pred[i]==1.0]
            x_2 = [d[1] for i,d in enumerate(red_data) if y_pred[i]==0.0]
            y_2 = [d[0] for i,d in enumerate(red_data) if y_pred[i]==0.0]
            plt.plot(x_1, y_1, 'c.', x_2, y_2, 'y.')
        return confusion_matrix(y, y_pred)

    #[0,0]-->True Negatives
    #[0,1]-->False Positives
    #[1,0]-->False Negatives
    #[1,1]-->True Positives

    def prec_from_conf_mat(conf_mat):
        return round( (conf_mat[0,0]+conf_mat[1,1])/np.sum(conf_mat), 4)

    def auc_from_conf_mat(conf_mat):
        #True Positives / Positives
        TPR = conf_mat[1,1]/(conf_mat[1,1]+conf_mat[1,0])
        #False Positives / Negatives
        FPR = conf_mat[0,1]/(conf_mat[0,0]+conf_mat[0,1])
        return round( (1.0+TPR-FPR)/2.0, 4)

    def f1_from_conf_mat(conf_mat):
        return round( 2*conf_mat[1,1]/(2*conf_mat[1,1]+
                conf_mat[0,1]+conf_mat[1,0]), 4)
```

En el código anterior, `test_model` calcula la matriz de confusión del modelo generado a partir del estimador pasado como parámetro y entrenado con los conjuntos X, de instancias, e y de etiquetas, usando validación cruzada para el número de particiones que se le indiquen (por defecto 5). Si se le especifica, visualiza un gráfico con los puntos del dataset reducidos a dos dimensiones (usando un reductor tipo PCA) y los colorea de azul o amarillo dependiendo de la predicción. Las otras tres funciones calculan la precisión, el AUC y la puntuación F1, respectivamente.

```
    #Parámetros: lista de estimadores, lista con sus respectivos nombres,
    #             conjuntos de características, vector de valores objetivo.
    #Devuelve: un dataframe con el nombre, precisión, AUC y F-score de cada modelo generado
    #          usando los estimadores pasados como parámetro
    def test_todos(estimadores, nombres, X, y):
        assert len(estimadores)==len(nombres)
        tabla = []
        for estim,nombre in zip(estimadores, nombres):
            conf_mat = test_model(estim, X, y)
            prec = prec_from_conf_mat(conf_mat)
            auc = auc_from_conf_mat(conf_mat)
            f1 = f1_from_conf_mat(conf_mat)
            tabla.append([prec, auc, f1])
        return pd.DataFrame(tabla, index=nombres, columns=['Precisión', 'AUC', 'F1-Score'])

    def normalizar_dataset(data):
        minimo = min(data.ravel())
        maximo = max(data.ravel())
        rango = maximo-minimo
        return (data-minimo)/rango

    char = data.values[:,0:5]
    target = data.values[:,5]
    red_char = reducer.transform(data.values[:,0:5])  #Reduced characteristics
    norm_char = normalizar_dataset(char)
```

`test_todos` utiliza `test_model` para calcular la matriz de confusión de cada modelo y, a raíz de ella, el resto de métricas. `normalizar_dataset` nos servirá para normalizar el conjunto de datos y comparar resultados con el entrenamiento de los modelos usando datos sin normalizar. En las últimas cuatro líneas definimos las matrices de características para cada uno de los procesados del conjunto de datos y el vector de etiquetas.

Finalmente, definimos el conjunto de características en el que convertimos los parámetros cualitativos a arrays binarios (One Hot Arrays).

```
categorical_features = ['bi-rads', 'shape', 'margin', 'density']
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder())])

ohe = ColumnTransformer(transformers=[('cat', categorical_transformer, categorical_features)])

char_columns = ['bi-rads', 'age', 'shape', 'margin', 'density']
onehot_char = ohe.fit_transform(data[char_columns]).toarray()
```

Árbol de decisión

Código

```
from sklearn.tree import DecisionTreeClassifier
conf_mat = test_model(DecisionTreeClassifier(max_depth=3),
                      char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 465 & 51 \\ 107 & 338 \end{pmatrix}$$

Precisión: 0.8356
AUC: 0.8304
F1-Scores: 0.8106

El dataset normalizado obtiene los mismos resultados, puesto que la segmentación que hace `DecisionTreeClassif` del espacio no depende de la escala. El dataset reducido a sus 3 mejores características seleccionadas por el test χ^2 obtiene los siguientes resultados:

$$CM = \begin{pmatrix} 387 & 129 \\ 67 & 378 \end{pmatrix}$$

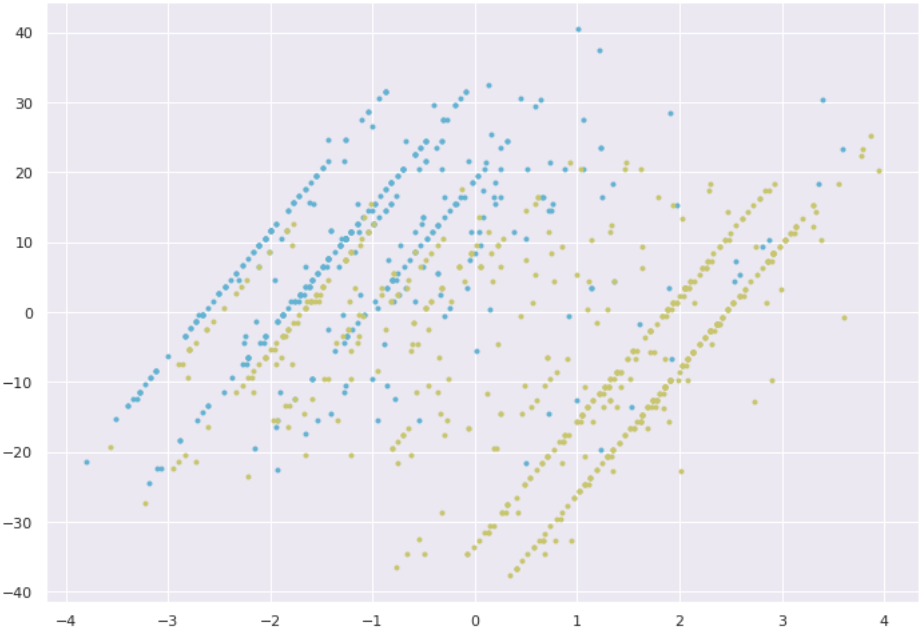
Precisión: 0.796
AUC: 0.7997
F1-Scores: 0.7941

Los resultados para el modelo usando la codificación OneHot son:

$$CM = \begin{pmatrix} 416 & 100 \\ 79 & 366 \end{pmatrix}$$

Precisión: 0.8137
AUC: 0.8143
F1-Scores: 0.8035

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



AdaBoost

Código

```
from sklearn.ensemble import AdaBoostClassifier
conf_mat = test_model(AdaBoostClassifier(learning_rate=0.1, n_estimators=30),
                char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 462 & 54 \\ 102 & 343 \end{pmatrix}$$

Precisión: 0.8377
AUC: 0.8331
F1-Scores: 0.8147

El dataset normalizado obtiene los mismos resultados, puesto que la segmentación que hace `AdaBoost` del espacio no depende de la escala. La versión reducida tiene los resultados siguientes:

$$CM = \begin{pmatrix} 411 & 105 \\ 83 & 362 \end{pmatrix}$$

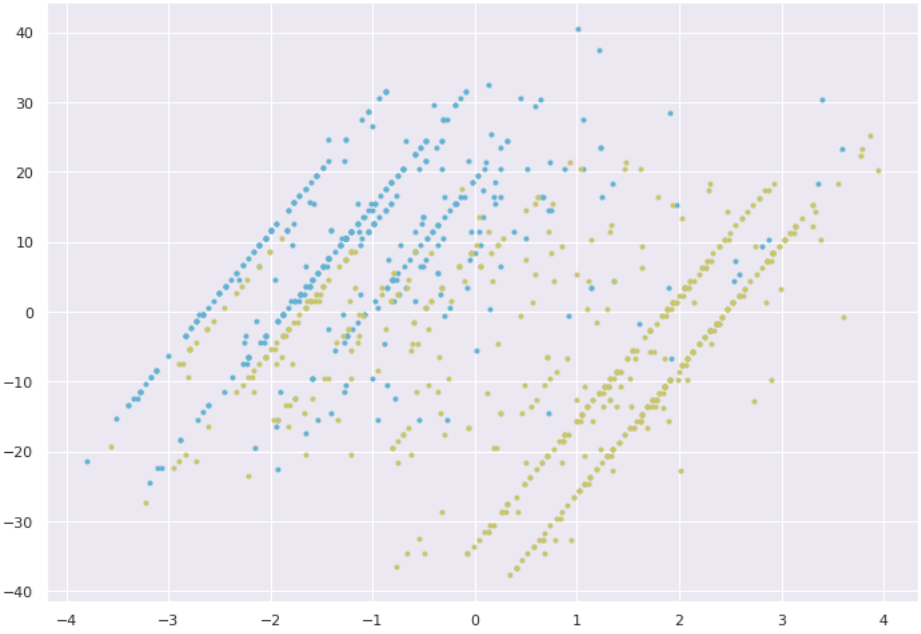
Precisión: 0.8044
AUC: 0.805
F1-Scores: 0.7939

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 435 & 81 \\ 101 & 344 \end{pmatrix}$$

Precisión: 0.8106
AUC: 0.808
F1-Scores: 0.7908

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



Regresión Logística

Código

```
from sklearn.linear_model import LogisticRegression
conf_mat = test_model(LogisticRegression(random_state=1, C=0.1),
                char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 424 & 92 \\ 76 & 369 \end{pmatrix}$$

Precisión: 0.8252
AUC: 0.8255
F1-Scores: 0.8146

Con los datos normalizados:

$$CM = \begin{pmatrix} 435 & 81 \\ 226 & 219 \end{pmatrix}$$

Precisión: 0.6805
AUC: 0.6676
F1-Scores: 0.5879

Con los datos reducidos obtenemos:

$$CM = \begin{pmatrix} 397 & 119 \\ 77 & 368 \end{pmatrix}$$

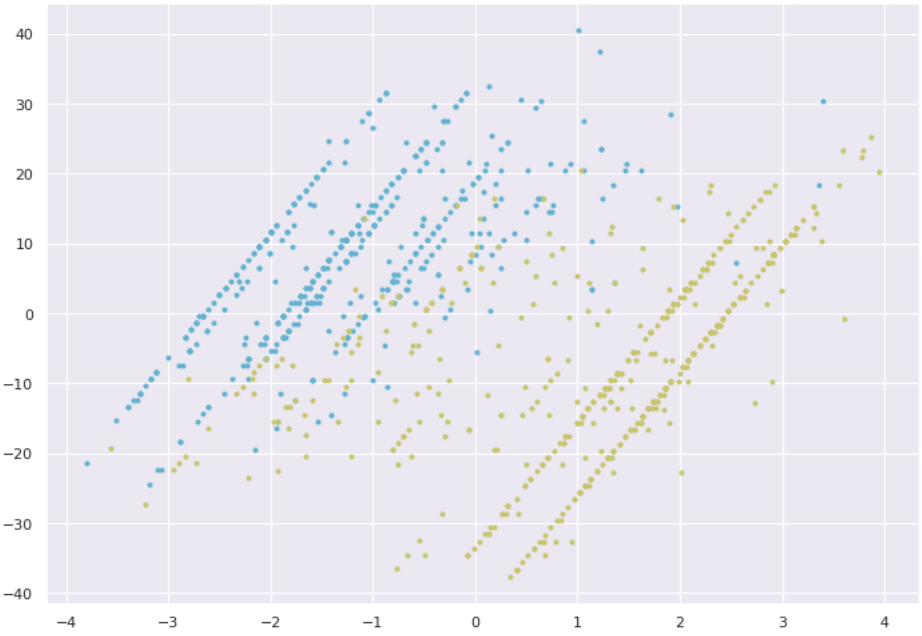
Precisión: 0.796
AUC: 0.7982
F1-Scores: 0.7897

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 427 & 89 \\ 91 & 354 \end{pmatrix}$$

Precisión: 0.8127
AUC: 0.8115
F1-Scores: 0.7973

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



Random Forest

Código

```
from sklearn.ensemble import RandomForestClassifier
conf_mat = test_model(RandomForestClassifier(max_depth=4, random_state=1),
                char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 442 & 74 \\ 80 & 365 \end{pmatrix}$$

Precisión: 0.8398
AUC: 0.8384
F1-Scores: 0.8258

Tal como ocurría con los árboles de decisión sencillos, la normalización del dataset no afecta al Random Forest. Con los datos reducidos obtenemos:

$$CM = \begin{pmatrix} 402 & 114 \\ 80 & 365 \end{pmatrix}$$

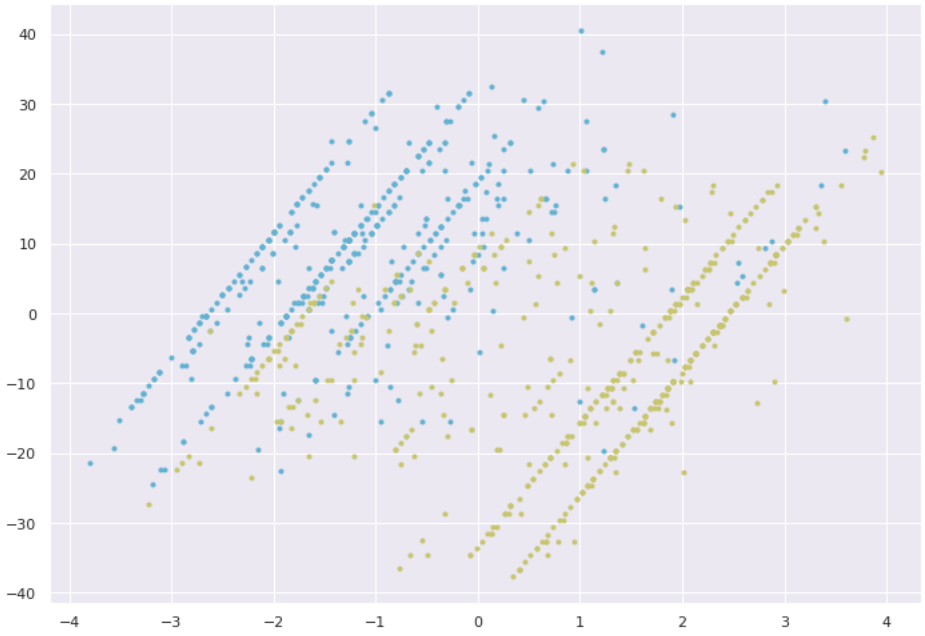
Precisión: 0.7981
AUC: 0.7996
F1-Scores: 0.0.79

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 424 & 92 \\ 83 & 362 \end{pmatrix}$$

Precisión: 0.8179
AUC: 0.8176
F1-Scores: 0.8053

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



Red Neuronal

Código

```
from sklearn.neural_network import MLPClassifier
conf_mat = test_model(MLPClassifier(hidden_layer_sizes=(25, 3), max_iter=10000, random_state=1),
                char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 408 & 108 \\ 70 & 375 \end{pmatrix}$$

Precisión: 0.8148
AUC: 0.8167
F1-Scores: 0.8082

Con los datos normalizados:

$$CM = \begin{pmatrix} 412 & 104 \\ 78 & 367 \end{pmatrix}$$

Precisión: 0.8106
AUC: 0.8116
F1-Scores: 0.8013

Con los datos reducidos obtenemos:

$$CM = \begin{pmatrix} 516 & 0 \\ 445 & 0 \end{pmatrix}$$

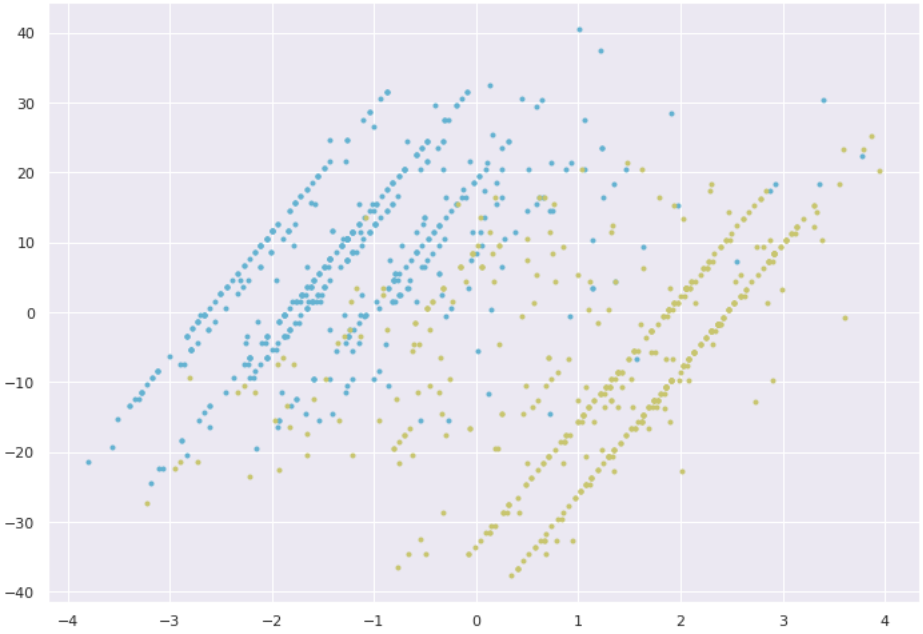
Precisión: 0.5369
AUC: 0.5
F1-Scores: 0.0

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 415 & 101 \\ 80 & 365 \end{pmatrix}$$

Precisión: 0.8117
AUC: 0.8122
F1-Scores: 0.8013

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



SVM

Código

```
from sklearn.svm import LinearSVC
conf_mat = test_model(LinearSVC(random_state=1, max_iter=300000),
                char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 423 & 93 \\ 77 & 368 \end{pmatrix}$$

Precisión: 0.8231
AUC: 0.8234
F1-Scores: 0.8124

Con los datos normalizados:

$$CM = \begin{pmatrix} 399 & 117 \\ 112 & 333 \end{pmatrix}$$

Precisión: 0.7617
AUC: 0.7608
F1-Scores: 0.7441

Con los datos reducidos obtenemos:

$$CM = \begin{pmatrix} 388 & 128 \\ 69 & 376 \end{pmatrix}$$

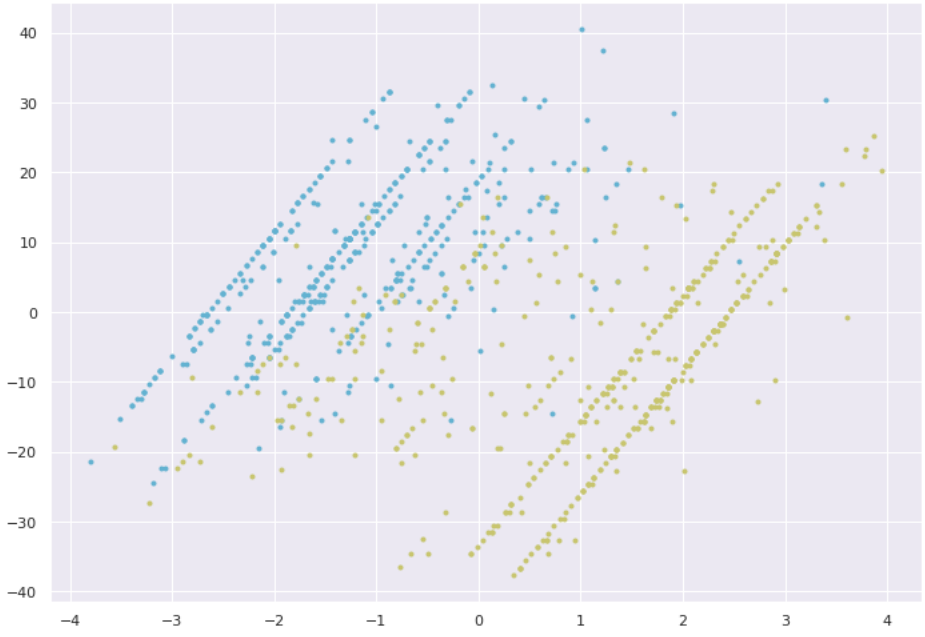
Precisión: 0.795
AUC: 0.7984
F1-Scores: 0.7924

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 440 & 76 \\ 99 & 346 \end{pmatrix}$$

Precisión: 0.8179
AUC: 0.8151
F1-Scores: 0.7982

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



Multiclasificador por Votación

Código

```
from sklearn.ensemble import VotingClassifier
clf1 = AdaBoostClassifier(learning_rate=0.1, n_estimators=30, random_state=1)
clf2 = LogisticRegression(random_state=1, C=0.1)
clf3 = RandomForestClassifier(max_depth=4, random_state=1)
eclf = VotingClassifier( estimators=[('ad', clf1), ('lr', clf2), ('rf', clf3)] )
conf_mat = test_model(eclf, char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 445 & 71 \\ 81 & 364 \end{pmatrix}$$

Precisión: 0.8418
AUC: 0.8402
F1-Scores: 0.8273

Con los datos normalizados:

$$CM = \begin{pmatrix} 460 & 56 \\ 95 & 350 \end{pmatrix}$$

Precisión: 0.8429
AUC: 0.839
F1-Scores: 0.8226

Con los datos reducidos obtenemos:

$$CM = \begin{pmatrix} 405 & 111 \\ 79 & 366 \end{pmatrix}$$

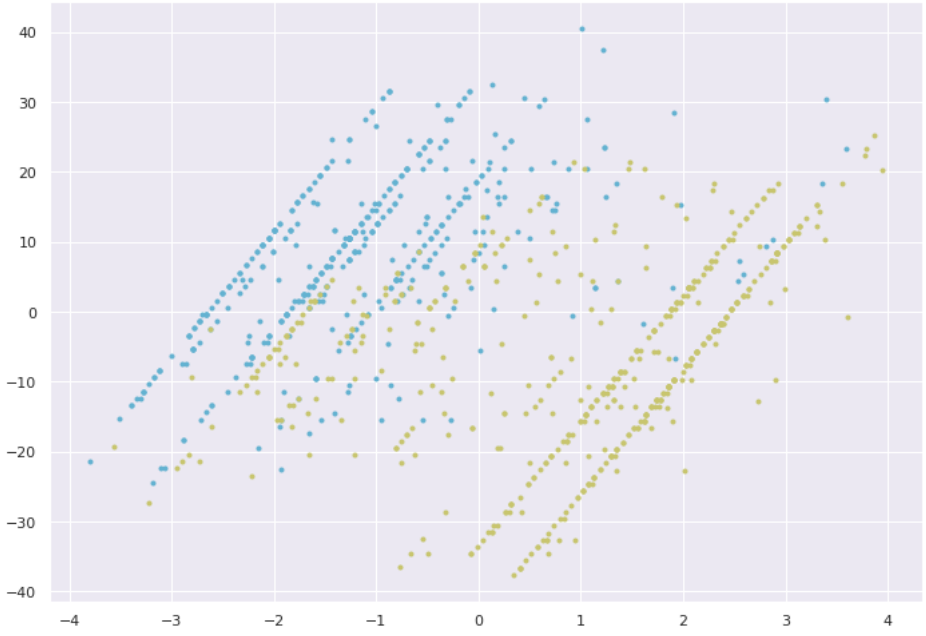
Precisión: 0.8023
AUC: 0.8037
F1-Scores: 0.7939

Con la codificación OneHot obtenemos:

$$CM = \begin{pmatrix} 424 & 92 \\ 84 & 361 \end{pmatrix}$$

Precisión: 0.8169
AUC: 0.8165
F1-Scores: 0.804

La clasificación hecha para el conjunto de datos original puede visualizarse en sus dos componentes principales de la siguiente forma:



5 Análisis de resultados

Mostramos los resultados medios tras 5 iteraciones de cada uno de los modelos. En cada ejecución se ha usado una semilla distinta de inicialización aleatoria.

Datos originales

	Precisión	AUC	F1-Score
Árbol de Decisión	0.8356	0.8304	0.8106
Random Forest	0.83496	0.8340400000000001	0.8217599999999999
AdaBoost	0.8377000000000001	0.8331	0.8147
Regresión Logística	0.8252	0.8255000000000001	0.8146000000000001
Red Neuronal	0.74672	0.74018	0.63172
SVM Lineal	0.8230999999999999	0.8234	0.8124
MultiClas por Votación	0.83808	0.83686	0.8243

Tras un primer vistazo a la tabla anterior, hay un resultado que destaca sobre los demás. La red neuronal ha conseguido, de media, resultados significativamente peores a los demás. Si recordamos los resultados de la red neuronal en el apartado 4, el motivo parece ser que la red neuronal, para los parámetros con los que se ha creado en esta práctica, es susceptible a converger rápidamente en óptimos locales. Este problema podría paliarse haciendo un estudio más exhaustivo de su convergencia y modificando sus parámetros, tal como se ha comentado en el apartado 3.

El siguiente aspecto que merece nuestro interés es el hecho de que el árbol de decisión, además de ser uno de los modelos más interpretables, compite en bondad de resultados con el resto de los métodos utilizados. En el apartado 6 explicaremos más a fondo que conclusiones obtenemos de esto de cara a la interpretación de los datos. Si bien no se han medido con precisión los tiempos de entrenamiento de los distintos modelos, para la profundidad del árbol utilizada se he observado una velocidad de entrenamiento muy superior al del resto de técnicas.

Otro detalle destacable es el hecho de que los multiclasificadores utilizados (RandomForest y Multiclasificador por Votación con AdaBoost, RandomForest y Regresión Logística) hayan conseguido unas precisiones de entre

las más altas, y las puntuaciones F1 más altas de entre todos los modelos testados. En un caso como el que nos ocupa en esta práctica, de clasificación de tumores, es especialmente importante evitar los falsos negativos que la medida F1 castiga especialmente. Haber obtenido las puntuaciones F1 más altas los convierte, posiblemente, en los mejores clasificadores para resolver este problema específico (al margen de los árboles de decisión, útiles por su interpretabilidad).

SVM y Regresión Logística no han logrados resultados especialmente buenos en este caso. SVM, de forma similar a la red neuronal, podría ser susceptible de un ajuste de parámetros más exhaustivo, pero debido a los largos tiempos de entrenamiento de este modelo realizar ese ajuste de forma efectiva para esta práctica era inabarcable. En el caso de la Regresión Logística, es posible que no exista una dependencia suficientemente fuerte entre las variables explicativas y la explicada, y por eso esta funciona peor que otros estimadores como el AdaBoost o los árboles de decisión, que aplican una separación geométrica del espacio (más sobre esto en el apartado sobre datos codificados).

Datos normalizados

	Precisión	AUC	F1-Score
Árbol de Decisión	0.8356	0.8304	0.8106
Random Forest	0.83496	0.8340400000000001	0.8217599999999999
AdaBoost	0.8377000000000001	0.8331	0.8147
Regresión Logística	0.6805	0.6676	0.5879
Red Neuronal	0.8152000000000001	0.8160000000000001	0.80558
SVM Lineal	0.7617	0.7608	0.7441
MultiClas por Votación	0.8408000000000001	0.8369199999999999	0.8202800000000001

Al normalizar el conjunto de datos, vemos un comportamiento heterogéneo en los resultados para distintos modelos. En el caso de los árboles, el Random Forest y AdaBoost, no sufren ningún cambio porque la segmentación espacial que realizan no depende de la escala del conjunto.

La Regresión Logística ha obtenido resultados mucho peores. Esto puede deberse a que este estimador regulariza los datos por defecto, además de que la escala del conjunto de datos sea de gran importancia en el cálculo de las probabilidades (la función logística no es lineal). Podría decirse que no tiene sentido normalizar el conjunto de datos si se desea entrenar este tipo de modelo.

La red neuronal ha mejorado sus resultados medios al normalizar. Esto puede deberse o bien a que se haya reducido la probabilidad de convergencia hacia óptimos locales en el espacio de soluciones para el dataset normalizado, o bien a que, para este caso particular, no se ha producido esa convergencia prematura. Sin un estudio más detallado no podemos concluir nada con mayor seguridad.

El modelo SVM ha obtenido resultados peores a los análogos para el dataset sin normalizar, lo cual resulta sorprendente en un primer momento porque la normalización es un paso muy recomendado para el uso de la SVM en la literatura del aprendizaje estadístico. La causa puede deberse al hecho de que hemos utilizado los mismos hiperparámetros para este caso que para el anterior y, a falta de un estudio más profundo de los mismos, resulte que estos fuesen adecuados para dicho dataset y no para el actual.

El Clasificador por Votación ha obtenido resultados incluso mejores a los anteriores. Recordemos que este multiclasificador lo hemos creado a partir de un Random Forest, un AdaBoost y un modelo de Regresión Logística, de los cuales el único que ha cambiado sus resultados para el nuevo dataset, y a peor, es la Regresión Logística. Los resultados pueden deberse a la forma del dataset que estamos tratando y a las soluciones específicas de los modelos escogidos. Pudiera ser también que la Regresión Logística, que tenía antes resultados ligeramente peores a los otros dos, ahora realice una clasificación peor pero más sencilla, eliminando ruido en las zonas que puedan clasificarse claramente como 'malignas' o claramente como 'benignas'.

Datos reducidos mediante Selección de Características

	Precisión	AUC	F1-Score
Árbol de Decisión	0.796	0.7997	0.7941
Random Forest	0.79958	0.80114	0.7916
AdaBoost	0.8044	0.805	0.7939
Regresión Logística	0.796	0.7982	0.7897
Red Neuronal	0.74192	0.73614	0.6290600000000001
SVM Lineal	0.795	0.7984	0.7924
MultiClas por Votación	0.8043800000000001	0.80556	0.7954800000000001

Para el conjunto de datos reducido los resultados son, en todos los casos, peores. La explicación es sencilla: si bien las componentes eliminadas no aportaban una gran cantidad de información al conjunto, esta era información útil que no ha merecido la pena eliminar, debido a que la baja dimensionalidad que ya presentaba el dataset no suponía ningún problema para el aprendizaje de nuestros modelos.

Los multiclasificadores, que antes poseían las puntuaciones F1 más altas, ahora se encuentran al mismo nivel que los clasificadores simples (ligeramente por encima del 0.79). Esto podría señalarnos que los multiclasificadores mejoraban la robustez ante falsos negativos gracias a la información aprendida de estas variables que hemos eliminado.

Datos codificados a OneHot

	Precisión	AUC	F1-Score
Árbol de Decisión	0.8137	0.8143	0.8035
Random Forest	0.820775	0.820875	0.809475
AdaBoost	0.8106	0.808	0.7908
Regresión Logística	0.8127	0.8115	0.7973
Red Neuronal	0.807225	0.8063499999999999	0.792375
SVM Lineal	0.8179	0.8151	0.7982
MultiClas por Votación	0.8176499999999999	0.81725	0.8049

Aunque la codificación binaria de las variables categóricas tenía sentido en primer lugar, observamos que los resultados son peores que los del dataset original. La única excepción es la red neuronal aunque, como ya hemos explicado para el dataset normalizado, no podemos asegurar que la transformación hecha realmente favorezca la convergencia adecuada de la red o solo sea fruto del azar.

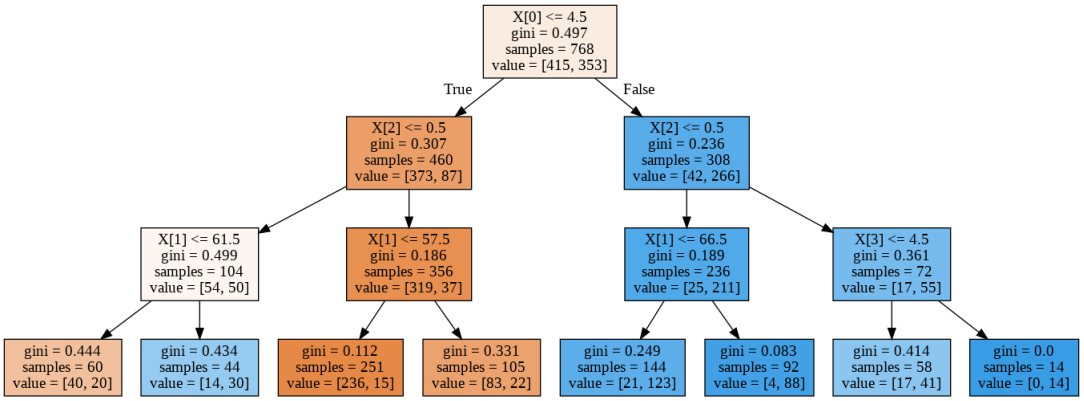
Una razón de que los resultados empeoren al codificar los datos ha podido ser el ajuste de parámetros. Tal vez un ajuste distinto hubiese sido más beneficioso para procesar datos mixtos (numéricos y categóricos) como los que hemos utilizado en este caso.

Otra causa puede deducirse de la visualización que hicimos de la clasificación en las componentes principales para el apartado 2. En dicha visualización observamos que la clasificación de cada instancia sigue, en gran medida, un patrón geométrico (los casos de tumores malignos se posicionan, mayormente, en las columnas de la izquierda, y los benignos en las de la derecha). Esto puede indicar que, aunque los parámetros sean en su mayoría cualitativos, el orden que se les ha dado al etiquetarlos ha sido, en la práctica, beneficioso para su clasificación.

6 Interpretación de los datos

Vamos a proceder a argumentar que características de entre las estudiadas identifican mejor a los tumores benignos y malignos, apoyados en el árbol de decisión implementado y en los distintos gráficos mostrados.

Volvamos a visualizar el árbol de decisión:



De acuerdo al árbol anterior, podemos clasificar un tumor mirando primero su característica 0 (BI-RADS), luego la 2 (Shape) y luego la 1 (Age) o la 3 (Margin). Claramente, como que la precisión de este modelo no ha sido total no podemos dar por hecho que estas variables sean las únicas importantes en la clasificación del tumor. Sin embargo, esta división del dataset tiene sentido si miramos los gráficos de la figura 1 (apartado 2).

- Un valor de BI-RADS de 5 señala que el tumor será probablemente maligno.
- Tanto si la forma es irregular, como si es redonda u ovalada, la forma indica con alta probabilidad la clase del tumor.
- En el gráfico de la edad vemos claramente que los tumores en personas jóvenes son más propensos a ser benignos, luego esta también parece ser una característica útil.
- Si el margen de masa vale 5 (tal como se comprueba en la primera hoja del árbol empezando por la derecha) el tumor será, probablemente, maligno.

La densidad es, con diferencia, la característica menos necesaria. No sólamente porque presente un valor de 3 en la mayor parte de las filas de la tabla de datos, sino porque para casi todos sus posibles valores las cantidades de tumores etiquetados como benignos y malignos están relativamente equilibradas.

7 Contenido adicional

Análisis Factorial

Además del Análisis de Componentes Principales, se ha realizado un Análisis Factorial para estudiar la dimensionalidad del espacio en función de sus variables ocultas. Algunas de las instancias del dataset de mamografías, transformadas a una composición de estas variables ocultas, las mostramos a continuación:

$$\begin{bmatrix} 8.015e-01 & -6.762e-01 & -6.694e-02 & 0 & 0 \\ -8.746e-01 & 9.588e-01 & 1.555e-01 & 0 & 0 \\ 1.858e-01 & -1.269 & -3.691e-02 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 5.961e-01 & -9.347e-01 & 5.345e-02 & 0 & 0 \\ 7.361e-01 & -1.039 & -3.383e-02 & 0 & 0 \\ 4.475e-01 & 4.503e-02 & -8.019e-04 & 0 & 0 \end{bmatrix}$$

Los resultados refuerzan las conclusiones obtenidas a partir de la selección de características. No todas las variables son necesarias, y la información del conjunto puede representarse en un espacio de menor dimensión.

Modelos adicionales: Naïve Bayes

Código

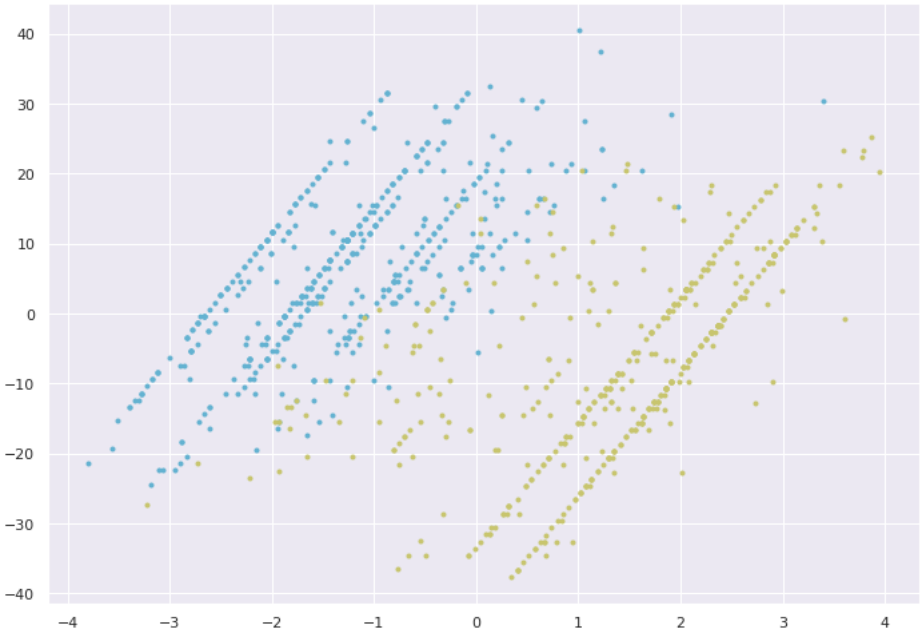
```
from sklearn.naive_bayes import GaussianNB
conf_mat = test_model(GaussianNB(), char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 403 & 113 \\ 66 & 379 \end{pmatrix}$$

Precisión: 0.8137
AUC: 0.8163
F1-Scores: 0.809

Visualización



Modelos Adicionales: k-NN

Código

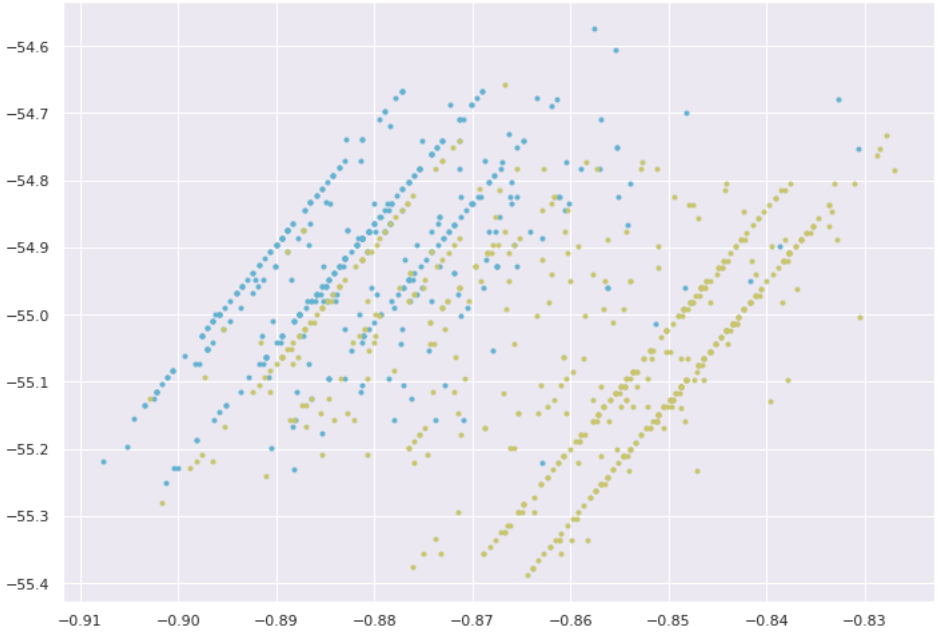
```
from sklearn.neighbors import KNeighborsClassifier
conf_mat = test_model(KNeighborsClassifier(5, metric='hamming', weights='uniform'),
norm_char, target, pca, show_plot=True)
```

Resultados

$$CM = \begin{pmatrix} 449 & 67 \\ 114 & 331 \end{pmatrix}$$

Precisión: 0.8117
AUC: 0.807
F1-Scores: 0.7853

Visualización



8 Bibliografía

- Pandas
- Matplotlib
- Seaborn
- NumPy
- Scikit-learn
- Boosting (Wikipedia)
- Análisis Factorial (Guía de usuario de Scikit)