LABORATORIO – APRENDIZAJE SUPERVISADO

1. Introducción

El aprendizaje automático supervisado desempeña un papel fundamental en la resolución de problemas de clasificación en diversas disciplinas, tan variadas como informática, biología, economía, ciencias de la salud… En esta actividad se nos propone implementar un estudio con técnicas de aprendizaje para abordar un problema específico a nuestra elección, siempre que cumpla unos requisitos. La elección de un conjunto de datos apropiado y la discusión y comparación de los resultados obtenidos por las diferentes técnicas nos ayudará a profundizar más en la aplicación práctica de estos métodos, así como a afianzar los conocimientos adquiridos a lo largo del bloque de Aprendizaje Supervisado.

El problema consiste en un conjunto de datos acerca de imágenes de judías secas que incluyen variables acerca de su estructura y una etiqueta con la clase a la que pertenecen, dentro de las 7 que aparecen en el dataset. Nuestro objetivo será diseñar un modelo que sea capaz de predecir la clase a la que pertenece cada una de las judías a partir de sus características, por lo que se trata de un problema de clasificación.

Al obtener un modelo que sea capaz de predecir el tipo de judía con una alta tasa de acierto, se podría emplear en aquellos sectores donde se trabaje a la vez con judías de distintos tipos, tales como agricultura, comercio..., con el fin de mejorar la eficiencia y rapidez del proceso al ser capaces de automatizar el proceso.

Por último, hay que señalar que este informe está organizado en distintos apartados que incluyen información acerca del problema, la metodología que hemos seguido para obtener los distintos modelos, los resultados obtenidos, una discusión de los resultados y las conclusiones obtenidas.

2. Descripción del problema

Disponemos de un dataset con datos acerca de 13.611 imágenes de judías diferentes, tomadas con una cámara de alta resolución. De cada una de ellas disponemos de 16 atributos, así como una etiqueta que nos indica la variedad a la que pertenece cada una de las judías (DERMASON, SIRA, SEKER, HOROZ , CALI, BARBUNYA, BOMBAY). Los atributos conocidos son:

- **Area**: El área de una judía y el número de píxeles dentro de sus límites.
- **Perimeter**: La circunferencia de la judía se define como la longitud de su borde.
- **MajorAxisLength**: Distancia entre los extremos de la línea más larga que se puede trazar a partir de una judía.
- **MinorAxisLength**: La línea más larga que se puede trazar desde la judía estando perpendicular al eje principal.
- **AspectRation**: Define la relación entre MajorAxisLength y MinorAxisLength.
- **Eccentricity**: Excentricidad de la elipse que tiene los mismos momentos que la región.
- **ConvexArea**: Número de píxeles del polígono convexo más pequeño que puede contener el área de una judía.
- **EquivDiameter**: Diámetro de un círculo que tiene la misma superficie que una judía.

- **Extent**: La relación entre los píxeles del cuadro delimitador y el área de la judía.
- **Solidity**: También llamada convexidad. La relación entre los píxeles de la cáscara convexa y los que se encuentran en las judías.
- **Roundness**: Calculada con la siguiente fórmula: (4piArea)/(Perimeter^2)
- **Compactness:** Mide la redondez de un objeto: EquivDiameter/MajorAxisLength
- **ShapeFactor1**: Factor de forma 1
- **ShapeFactor2**: Factor de forma 2
- **ShapeFactor3**: Factor de forma 3
- **ShapeFactor4**: Factor de forma 4

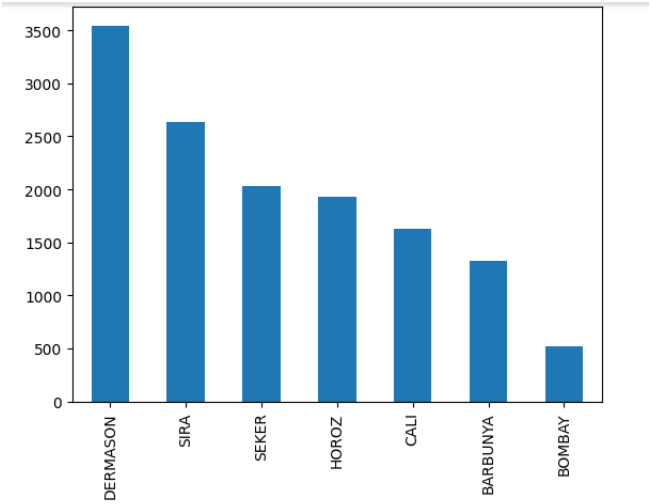
Todos los datos son de tipo numérico y, al visualizarlos, tienen la siguiente apariencia:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	0.913358
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272750	0.783968	0.984986	0.887034	0.953861
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	0.908774
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	0.928329
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	0.970516

También podemos observar sus medidas estadísticas:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	r
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	136
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.200206	253.064220	0.749733	0.987143	
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.915817	59.177120	0.049086	0.004660	
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.000000	161.243764	0.555315	0.919246	
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928	36714.500000	215.068003	0.718634	0.985670	
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.000000	238.438026	0.759859	0.988283	
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466	62294.000000	279.446467	0.786851	0.990013	
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.000000	569.374358	0.866195	0.994677	

Y la distribución de los datos de acuerdo con la clase a la que pertenecen:



Resaltar que se trata de una base de datos muy limpia y sin valores ausentes, por lo que resulta muy fácil trabajar con ella. Esto se debe a que el preprocesamiento de los datos necesario es mínimo, lo que nos permite pasar directamente a definir los modelos.

Por último, señalar que este dataset cumple con los requisitos que se planteaban en la actividad, pues tiene más de 15 variables que se emplearán en cada uno de los modelos y más de 100 instancias.

3. Metodología

Podemos distribuir nuestro estudio en 5 apartados principales: un breve preprocesamiento de los datos para asegurarnos que cumplen las condiciones necesarias para generar un modelo a partir de ellos y la aplicación de las 4 técnicas de aprendizaje automático vistas en clase (KNN, arboles de decisión, regresión logística y SVM). A continuación, se desarrolla más en profundidad cada uno de estos apartados:

- **Preprocesamiento**

El primer paso ha sido asegurarnos de que no exista ninguna fila completamente vacía, así como analizar la presencia de datos nulos en alguna de las columnas. De acuerdo con la descripción del dataset, no tenía lugar ninguno de los dos sucesos, y con esta rápida comprobación, nos aseguramos de que efectivamente es así.

El siguiente paso ha sido convertir a dato numérico las etiquetas de cada una de las clases. Para ello, definimos un *LabelEncoder* y lo aplicamos a la última columna del dataset ('Class'). Como esta función asigna un número en función del orden de aparición de los datos, guardamos en un diccionario cada uno de los números con la clave a la que hacen referencia. De esta forma, conservamos esta información y podemos acceder a ella de forma rápida en caso de necesitarla más adelante.

A continuación, dividimos el conjunto de datos en conjunto x (variables) e y (etiquetas) mediante la función *iloc()* de *pandas*. Una vez hecho esto, obtenemos los conjuntos de entrenamiento y de test, con una proporción del 80% de los datos disponibles destinados a entrenamiento y el 20% a test. Como semilla de aleatoriedad usamos el 42, para que el experimento pueda ser replicado más fácilmente en el futuro. Al hacer esto, nos damos cuenta de que las variables *ytrain* e *ytest* siguen siendo de tipo *dataframe*, así que las convertimos a *array*.

Finalmente, escalamos los datos mediante un *StandardScaler.* Aunque este paso no es necesario para todas las técnicas que vamos a aplicar, si lo es para algunas de ellas (KNN, Regresión Logística y SVM), así que lo realizamos en primer lugar para no tener problemas en el futuro.

K-Nearest Neighbor

En primer lugar, teniendo en cuenta que nuestro objetivo es encontrar el mejor número de vecinos (k) que permita clasificar nuestros datos y, para ello, emplearemos la técnica de la validación cruzada, hemos decidido buscar un k dentro de un rango entre 1 y 50 y emplear como métrica la distancia euclídea en nuestro clasificador. Después, hemos definido un *GridSearchCV* con 10 *folds*, que usará validación cruzada para obtener el mejor parámetro en base al *accuracy* de cada uno de los modelos. Lo entrenamos con los datos de entrenamiento y obtenemos que el mejor k es 10.

Ahora, definimos nuestro modelo empleando este parámetro y lo entrenamos con los datos de entrenamiento, obteniendo un *accuracy* de entrenamiento de alrededor de 0.934. También realizamos predicciones sobre el conjunto de test, que ha permanecido independiente durante todo el proceso,

y las comparamos con los datos disponibles. Obtenemos un accuracy de test de 0.925, lo cual es un buen indicador de que nuestro modelo funciona correctamente y no se ha producido *overfitting.* Como los valores de precisión obtenidos tanto en entrenamiento como en test son altos, podemos descartar que se haya producido *underfitting*.

Por último, mostramos las matrices de confusión.

Árboles de decisión

Comenzamos inicializando dos contadores en 0 que nos servirán posteriormente para obtener el accuracy del modelo de forma "manual", comparando el número de coincidencias entre las predicciones y los datos disponibles.

Como en el apartado anterior, comenzamos definiendo el rango de valores que pueden tomar los parámetros de nuestro modelo y que se comprobarán mediante validación cruzada para obtener los mejores. Los parámetros que se estudiarán serán: el número mínimo de muestras necesarias para estar en un nodo hoja (5 o 10), el número mínimo de muestras necesarias para dividir un nodo interno (5, 10 o 20), la máxima profundidad del árbol (2, 5, 10 o 100) y el criterio de medida (*entropy* que mide la ganancia de información de Shannon o *gini* que mide la impureza de Gini). Como antes, definimos un *GridSearchCV* con 10 folds y lo entrenamos usando el conjunto de datos de entrenamiento.

Obtenemos que el mejor modelo de árbol de decisión con estos datos tiene los siguientes parámetros: DecisionTreeClassifier(ccp_alpha=0, max_depth=10, min_samples_leaf=5, min_samples_split=20). Al realizar predicciones sobre el conjunto de entrenamiento, obtenemos un accuracy de entrenamiento de aproximadamente 0.941. Realizamos predicciones sobre el conjunto de test, que ha permanecido independiente durante todo el proceso, y obtenemos un accuracy de test de aproximadamente 0.911. Con estos resultados, podemos descartar que se hayan producido tanto *overfitting* como *underfitting*.

Por último, mostramos las matrices de confusión y el árbol obtenido, cuyas reglas generadas interpretaremos más adelante.

Regresión Logística

Comenzamos definiendo el rango de valores que pueden tomar los parámetros de nuestro modelo y que se comprobarán mediante validación cruzada para obtener los mejores, al igual que en los apartados anteriores. Los parámetros que comprobaremos son: C, que indica la inversa de la intensidad de la regularización aplicada (0.001, 0.01, 0.1, 1, 10 o 100) y la penalización aplicada ('l1' o 'l2'). En un primer momento, también se quiso probar con distintos algoritmos para resolver el problema de optimización ('liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga'), pero esto incrementaba en gran medida el tiempo de computación por lo que se volvía inviable ejecutar el código cada vez que se reiniciaba el entorno de GoogleColab. Por ello, se decidió descartar esta búsqueda y emplear el algoritmo que se usa por defecto ('lbfgs').

Más adelante, definimos un *GridSearchCV* con 10 folds y este conjunto de parámetros, que se entrenará con los datos de entrenamiento y hará uso de validación cruzada para obtener los mejores parámetros para estos datos. Lo entrenamos usando el conjunto de datos de entrenamiento. Durante el proceso de entrenamiento se muestran una serie de warnings debido a que con algunos parámetros

se alcanza el número máximo de iteraciones antes de que la función converja, pero al final obtenemos el modelo, cuyos mejores parámetros son: LogisticRegression(C=100).

Inicializamos dos parámetros en 0 que nos servirán para obtener el accuracy de los conjuntos de entrenamiento y test. Realizamos predicciones sobre el conjunto de entrenamiento y obtenemos un accuracy de aproximadamente 0.924. Mismo procedimiento para el conjunto de test donde obtenemos un accuaracy de aproximadamente 0.926.

Estos resultados nos permiten descartar que existan problemas de *overfitting* o *underfitting*. Sin embargo, es bastante destacable e inusual que se obtenga un accuracy superior en test que en entrenamiento, sobre todo con el volumen de datos que disponemos. Como la diferencia es mínima y, al revisar los parámetros de ajuste del modelo y el proceso de división de los datos, no se encuentra ningún error, se supondrá que ha sido fruto del azar.

Finalmente, se imprimen las matrices de confusión de ambos conjuntos.

SVMs

Comenzamos definiendo el rango de valores que pueden tomar los parámetros de nuestro modelo y que se comprobarán mediante validación cruzada. Los parámetros que comprobaremos son: C, que indica la inversa de la intensidad de la regularización aplicada (0.001, 0.01, 0.1, 1, 10 o 100), el tipo de kernel usado por el algoritmo ('linear', 'rbf' o 'poly') y el coeficiente del kernel usado (‘scale’ o ‘auto’).

Definimos un *GridSearchCV* con 10 folds y este conjunto de parámetros, que se entrenará con los datos de entrenamiento y hará uso de validación cruzada para obtener los mejores parámetros para estos datos. Lo entrenamos usando el conjunto de datos de entrenamiento y obtenemos los siguientes parámetros: SVC(C=100).

Como hemos hecho anteriormente, inicializamos dos parámetros en 0 y realizamos predicciones tanto en el conjunto de entrenamiento, donde obtenemos un accuracy de aproximadamente 0.943, como en el conjunto de test, que se ha mantenido independiente, donde obtenemos un accuracy de aproximadamente 0.929.

Estos resultados nos permiten descartar que existan problemas de *overfitting* o *underfitting*. Para finalizar, mostramos las matrices de confusión.

4. Resultados

Una vez obtenidos los distintos modelos y medido su precisión, los ordenamos en función de su accuracy en test, ya que se considera en este caso se prueba el modelo con un conjunto de datos completamente independiente al utilizado para obtenerlo, por lo que consideramos que es una prueba más realista de la eficacia del modelo :

Técnica empleada para el modelo	Accuracy (entrenamiento)	Accuracy (test)
SVMs	0.9439750183688465	0.9291222915901579
Regresión logística	0.9243203526818515	0.9265515975027543

KNN	0.9347905951506246	0.9250826294528094
Árboles de decisión	0.9416789125642909	0.9111274329783328

Además, mostramos las matrices de confusión para cada uno de los modelos, así como el *plot* del árbol de decisión:

KNN

```
----------------Train Confusion Matrix------------------
[[ 977    0   50    0    4    9   21]
 [   0  405    0    0    0    0    0]
 [  23    0 1261    0   17    2   10]
 [   0    0    0 2714    3   37  121]
 [   1    0   24   10 1450    0   35]
 [   3    0    0   35    1 1536   39]
 [   6    0    5  205   27   22 1835]]
              precision    recall  f1-score   support

           0       0.97      0.92      0.94      1061
           1       1.00      1.00      1.00       405
           2       0.94      0.96      0.95      1313
           3       0.92      0.94      0.93      2875
           4       0.97      0.95      0.96      1520
           5       0.96      0.95      0.95      1614
           6       0.89      0.87      0.88      2100

    accuracy                           0.93     10888
   macro avg       0.95      0.94      0.95     10888
weighted avg       0.93      0.93      0.93     10888
```

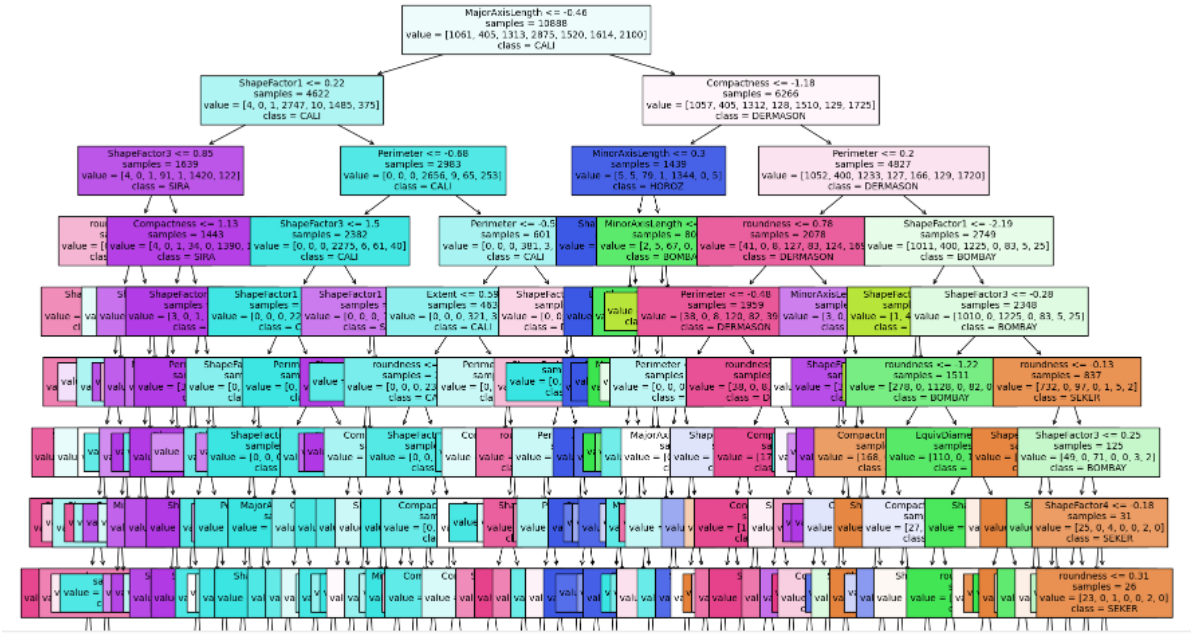
```
----------------Test Confusion Matrix------------------
[[234   0  19   0   0   1   7]
 [  0 117   0   0   0   0   0]
 [ 10   0 301   0   4   1   1]
 [  0   0   0 618   0   7  46]
 [  1   0   8   4 388   0   7]
 [  4   0   0   8   0 392   9]
 [  1   0   0  59   5   2 469]]
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       261
           1       1.00      1.00      1.00       117
           2       0.92      0.95      0.93       317
           3       0.90      0.92      0.91       671
           4       0.98      0.95      0.96       408
           5       0.97      0.95      0.96       413
           6       0.87      0.88      0.87       536

    accuracy                           0.93      2723
   macro avg       0.94      0.93      0.94      2723
weighted avg       0.93      0.93      0.93      2723
```

Árboles de decisión

```
----------------Train Confusion Matrix------------------
[[ 973    1   51    0    8    8   20]
 [   0  405    0    0    0    0    0]
 [  30    0 1257    0   18    2    6]
 [   0    0    0 2736    4   23  112]
 [   5    0   29   12 1451    0   23]
 [   6    0    1   30    0 1543   34]
 [   3    0   14  144   32   19 1888]]
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      1061
           1       1.00      1.00      1.00       405
           2       0.93      0.96      0.94      1313
           3       0.94      0.95      0.94      2875
           4       0.96      0.95      0.96      1520
           5       0.97      0.96      0.96      1614
           6       0.91      0.90      0.90      2100

    accuracy                           0.94     10888
   macro avg       0.95      0.95      0.95     10888
weighted avg       0.94      0.94      0.94     10888
```

```
----------------Test Confusion Matrix------------------
[[225   0  23   0   2   1  10]
 [  0 117   0   0   0   0   0]
 [ 12   0 300   0   4   0   1]
 [  0   0   0 612   1   6  52]
 [  1   0  10   6 383   0   8]
 [  3   0   0  20   0 378  12]
 [  3   0   2  48   8   9 466]]
              precision    recall  f1-score   support

           0       0.92      0.86      0.89       261
           1       1.00      1.00      1.00       117
           2       0.90      0.95      0.92       317
           3       0.89      0.91      0.90       671
           4       0.96      0.94      0.95       408
           5       0.96      0.92      0.94       413
           6       0.85      0.87      0.86       536

    accuracy                           0.91      2723
   macro avg       0.93      0.92      0.92      2723
weighted avg       0.91      0.91      0.91      2723
```

Además, se incluye la representación gráfica del árbol de decisión. Como son demasiadas reglas, en el código también se incluyen las reglas en formato de texto:



Regresión logística

```
----------------Train Confusion Matrix------------------
[[ 963    0   59    0    4    8   27]
 [   0  405    0    0    0    0    0]
 [  36    0 1238    0   22    3   14]
 [   1    0    0 2645    5   40  184]
 [   2    0   25   14 1440    0   39]
 [  13    0    1   23    1 1527   49]
 [   3    0    9  184   31   27 1846]]
              precision    recall  f1-score   support

           0       0.95      0.91      0.93      1061
           1       1.00      1.00      1.00       405
           2       0.93      0.94      0.94      1313
           3       0.92      0.92      0.92      2875
           4       0.96      0.95      0.95      1520
           5       0.95      0.95      0.95      1614
           6       0.86      0.88      0.87      2100

    accuracy                           0.92     10888
   macro avg       0.94      0.93      0.94     10888
weighted avg       0.92      0.92      0.92     10888
```

```
----------------Test Confusion Matrix------------------
[[238   0  14   0   0   0   9]
 [  0 117   0   0   0   0   0]
 [ 12   0 299   0   4   1   1]
 [  0   0   0 607   1   5  58]
 [  1   0   3   3 391   0  10]
 [  8   0   0   8   0 387  10]
 [  0   0   2  40   6   4 484]]
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       261
           1       1.00      1.00      1.00       117
           2       0.94      0.94      0.94       317
           3       0.92      0.90      0.91       671
           4       0.97      0.96      0.97       408
           5       0.97      0.94      0.96       413
           6       0.85      0.90      0.87       536

    accuracy                           0.93      2723
   macro avg       0.94      0.94      0.94      2723
weighted avg       0.93      0.93      0.93      2723
```

SVMs

```
----------------Train Confusion Matrix------------------
[[1010    0   29    0    3    6   13]
 [   0  405    0    0    0    0    0]
 [  20    0 1272    0   11    3    7]
 [   0    0    0 2729    2   26  118]
 [   1    0   17   12 1460    0   30]
 [   2    0    0   33    0 1550   29]
 [   6    0    1  206   15   20 1852]]
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      1061
           1       1.00      1.00      1.00       405
           2       0.96      0.97      0.97      1313
           3       0.92      0.95      0.93      2875
           4       0.98      0.96      0.97      1520
           5       0.97      0.96      0.96      1614
           6       0.90      0.88      0.89      2100

    accuracy                           0.94     10888
   macro avg       0.96      0.95      0.96     10888
weighted avg       0.94      0.94      0.94     10888
```

```
----------------Test Confusion Matrix------------------
[[240   0  12   0   0   3   6]
 [  0 117   0   0   0   0   0]
 [ 13   0 298   0   4   1   1]
 [  0   0   0 623   1   5  42]
 [  3   0   7   5 388   0   5]
 [  2   0   0   8   0 395   8]
 [  2   0   0  53   6   6 469]]
              precision    recall  f1-score   support

           0       0.92      0.92      0.92       261
           1       1.00      1.00      1.00       117
           2       0.94      0.94      0.94       317
           3       0.90      0.93      0.92       671
           4       0.97      0.95      0.96       408
           5       0.96      0.96      0.96       413
           6       0.88      0.88      0.88       536

    accuracy                           0.93      2723
   macro avg       0.94      0.94      0.94      2723
weighted avg       0.93      0.93      0.93      2723
```

5. Discusión

Observando la tabla con los accuracy de cada uno de los modelos, vemos que todos ellos alcanzan un nivel se precisión muy alto (mayor a 0.9 en todos los casos), lo que podría traducirse en que todos ellos serían válidos para emplearse en procesos de clasificación de judías secas.

En una primera impresión, el modelo obtenido mediante la técnica de SVMs es el que nos proporciona una mayor precisión tanto en entrenamiento como en test (por milésimas en algunos casos), con un valor de accuracy en test de 0.9291222915901579, por lo que sería el mejor de todos los modelos obtenidos. Como aspectos negativos de este modelo, podemos reseñar que su costo computacional suele ser mayor que los otros durante la fase de entrenamiento, especialmente para conjuntos grandes de datos o cuando empleamos un kernel complejo. Esto podemos observarlo durante el entrenamiento de nuestro modelo, en la fase de validación cruzada, cuando tarda más de 10 minutos en ejecutarse. Este tiempo podría reducirse incluyendo un menor número de parámetros a analizar, o empleando los que se usan por defecto. Sin embargo, hacer esto podría suponer una perdida de precisión a costa de ahorrar tiempo, ya que podría existir algún parámetro que no se comprobase y que diera lugar a un mejor modelo. En general, una vez entrenado el modelo, el costo computacional en la fase de predicción suele ser bajo con esta técnica. Otro aspecto negativo es el tema de la interpretabilidad, es decir la facilidad con la que se puede entender y explicar el funcionamiento del modelo, ya que los modelos obtenidos mediante Regresión Logística o Árboles de Decisión suelen ser más interpretables que los obtenidos mediante SVMs o KNN.

El segundo modelo con mayor tasa de precisión durante la fase de predicción es el obtenido mediante Regresión Logística, con un valor menos de 3 centésimas inferior al obtenido mediante SVMs, por lo que podría darse que, si repitiésemos el experimento, al variar la división de los datos, pudiese llegar a superarlo en términos de precisión. Como puntos a favor de este modelo podemos destacar que su costo computacional tanto en entrenamiento como en predicción suele ser menor que el de SVMs o KNN. Aunque en nuestro caso si que es cierto que durante la validación cruzada tuvimos que descartar el uso de distintos algoritmos porque elevaba mucho el tiempo de computación. Además, los modelos obtenidos mediante Regresión Lineal suelen ser más interpretables que los obtenidos mediante otras técnicas, como se ha señalado anteriormente. Como aspectos negativos, podríamos señalar que esta técnica suele ser más sensible a los datos atípicos, por lo que en caso de que nuestro conjunto de datos no estuviese tan limpio y ordenado, exigiría un preprocesamiento más exhaustivo que con otras técnicas. Podría llegar a valorarse que sus ventajas con respecto a SVMs fueran suficientes para elegir este modelo como definitivo para resolver el problema. Una forma de mejorarlo pasaría por realizar validación cruzada con un conjunto más grande de parámetros, incluyendo los *solvers* ya que, aunque tomara más tiempo, podría darse el caso de que se encontrase un modelo con una precisión mayor.

El tercer modelo con mayor tasa de precisión durante la fase de predicción es el obtenido mediante KNN, con una tasa de 0.9250826294528094, que no dista mucho de la obtenida por el mejor modelo. Como aspectos positivos podemos señalar que es fácil de implementar ya que no requiere tantos parámetros como en el resto de las técnicas y que presenta robustez ante cambios en los datos, ya que no realiza suposiciones específicas sobre la estructura de los datos. Como aspectos negativos podemos señalar que ofrece un nivel de interpretabilidad menor al de otras técnicas y que su coste computacional puede resultar más alto durante la fase de predicción ya que debe calcular la distancia entre el elemento de prueba y todos los elementos de entrenamiento para determinar los k vecinos más cercanos. Considerando que su coste computacional es más alto que otros en la fase de predicción y teniendo una tasa ligeramente peor de acierto que ellos, se consideran razones suficientes para no elegir este modelo como el definitivo para solucionar el problema. Una forma de mejorarlo sería

probando a emplear otras distancias en vez de la euclídea (como la distancia manhattan). De esta forma podría darse el caso de que obtuviésemos un modelo más preciso. Aumentar el valor del rango de k más allá de 50 no parece que vaya a suponer una mejora, ya que el mejor lo obtiene en 10, por lo que podríamos llegar incluso a reducir ese rango de búsqueda con el objetivo de reducir el tiempo de computación.

Por último, tenemos el modelo obtenido mediante Árboles de Decisión, con una tasa de precisión de 0.9111274329783328, que, aunque sea la peor de las obtenidas, da lugar a un modelo completamente válido. Como aspectos positivos de este modelo podemos destacar su interpretabilidad, ya que incluso podemos dibujar el árbol generado y observar las reglas que se van construyendo y como nuestro modelo toma decisiones para llegar a generar predicciones. Además, también podríamos señalar que no necesita de un escalado anterior, por lo que podríamos emplear los datos tal cual se obtienen de la base de datos, reduciendo el tiempo del proceso de preprocesamiento de los datos, y que presenta robustez ante datos con ruido, sin que estos afecten significativamente a su rendimiento. Como aspectos negativos podemos señalar que su costo computacional es alto, sobre todo en la fase de entrenamiento, por lo que debemos definir una profundidad máxima que permita que nuestro árbol goce de un alto nivel de precisión pero que no suponga un costo computacional excesivo. En la fase de predicción, suelen ser más rápidos. Una forma de mejorarlo podría pasar por emplear técnicas de *pruning* para evitar el sobreajuste y reducir el coste computacional. Después de haber analizado sus aspectos positivos y negativos y teniendo en cuenta que su accuracy es ligeramente menor que la del mejor modelo, se considera que los modelos obtenidos por SVMs y Regresión Logística son mejores para resolver el problema planteado, por lo que se decide descartar este modelo.

Para finalizar, vamos a interpretar las reglas generadas por el árbol de decisión. Como se genera un gran número de reglas, sería muy complicado y extenso analizarlas una a una, así que analizaremos solamente algunas de ellas (rama de la izquierda de la regla inicial):

- **Regla Inicial:** Si MajorAxisLength es menor o igual a -0.46, entonces pasa a la siguiente condición.
- **Primera Ramificación:**
    - Si ShapeFactor1 es menor o igual a 0.22, entonces se realizan otras comprobaciones basadas en otras características.
    - Si ShapeFactor1 es mayor que 0.22, entonces se realizan otras comprobaciones basadas en otras características.

Un ejemplo de como el árbol decidiría como clasificar un nuevo elemento en base a sus características sería (suponiendo que MajorAxisLength < -0.46, ShapeFactor1 < 0.22, ShapeFactor3 < 0.85 y Roudness < 0.89): Si MajorAxisLength es menor o igual a -0.46 y ShapeFactor1 es menor o igual a 0.22, y ShapeFactor3 es menor o igual a 0.85, y roundness es menor o igual a 0.89, entonces se llega a una clasificación de la clase 6. Este procedimiento continuaría hasta alcanzar los últimos nodos de la rama del árbol.

6. Conclusiones

Para concluir con el estudio realizado, debemos tomar una decisión acerca de cual de los dos modelos que no hemos descartado en el apartado anterior nos quedamos para resolver el problema planteado. Tenemos el modelo obtenido mediante SVMs y el obtenido mediante Regresión Logística.

Tras analizar los pros y las contras de ambos modelos y considerando que el tamaño de nuestro conjunto de datos es moderado (en caso de ser muy grande si que tomaría ventaja el uso de SVMs), se considera que menos de tres milésimas de precisión no son suficientes para superar los beneficios que ofrece la Regresión Logística en términos de eficiencia computacional e interpretabilidad del modelo. Por lo tanto, para resolver el problema planteado de clasificación emplearíamos el modelo obtenido mediante Regresión Logística.

7. Referencias

- KOKLU, M. and OZKAN, I.A., (2020), Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques. Computers and Electronics in Agriculture, 174, 105507. DOI: https://doi.org/10.1016/j.compag.2020.105507
- Teoría y ejemplos de código en el Moodle de la asignatura proporcionados por el profesor Esteban García Cuesta