Tarea 2

Autor: Arturo Lazcano **Profesor:** Felipe Tobar
Auxiliares: Catherine Benavides, Camila Bergasa, Víctor Caro, Camilo Carvajal Reyes, Diego Cortez M., Stefano Schiappacasse

P1: Pregunta teórica

(a) Sea $H \subseteq \mathbb{R}^n$ el hiperplano de separación de las 2 clases. Sea $w \in \mathbb{R}^n$ el vector normal a H. Notemos que el hiperplano de separación tiene la forma $H = x \in \mathbb{R}^n | w^T x + b = 0$ donde $b/||w||$ es el *offset* del hiperplano desde el origen a lo largo del vector normal w. Así, suponiendo que las clases son $\{-1, 1\}$, podemos obtener la predicción de un dato x viendo si $w^T x + b$ es mayor o menor a 0 (asignándolo a la clase 1 o -1 respectivamente). A priori este problema de de encontrar (w, b) no tiene solución única pues $(\lambda w, \lambda b)$ también es solución para $\lambda > 0$. Es por esto que impondremos una restricción al problema: Llamemos x_+, x_- vectores soporte, lo cuales son aquellos más cercanos al hiperplano por cada clase (pueden no ser únicos). Así, se quiere que cada vector soporte pertenezca a su clase, es decir, $w^T x_+ + b = 1$ y $w^T x_- + b = -1$. Por otro lado, veamos que el ancho de margen está dado por
$m = \frac{1}{2}||\text{proy}_w(x_+ - x_-)||$
$= \frac{1}{2}||x_+ - x_-||(\frac{w^t(x_+ - x_-)}{||w|| \cdot ||x_+ - x_-||})$
$= \frac{1}{2||w||} w^T(x_+ - x_-)$
$= \frac{1}{2||w||} w^T(x_+ - x_-)((w^T x_+) - (w^T x_-))$
$= \frac{1}{2||w||}((1 - b) - (-1 - b)) = \frac{1}{||w||}$. Consideremos $y_i = 1$ si $w^T x + b \geq 1$ e $y_i = -1$ si $w^T x + b \leq -1$. Con esto, podemos formular el problema como

$$\begin{aligned} \max_{w,b} \quad & \frac{1}{||w||} \\ \text{s.a.} \quad & y_i(w^T x + b) \geq 1, \ i \in 1, ..., N \end{aligned} \tag{1}$$

Por temas de diferenciabilidad, consideremos el cuadrado del término $||w||$ y minimizaremos (notar que el término 1/2 no afecta esta minimización)

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.a.} \quad & y_i(w^T x + b) \geq 1, \ i \in \{1, ..., N\} \end{aligned} \tag{2}$$

Con esto, el lagrangiano viene dado por $L(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{N} \alpha_i (1 - y_i(w^T x_i + b))$. Luego, el lagrangiano del dual viene dado por $\theta(\alpha) = \inf_{w,b} L(w, b, \alpha)$, y como L es convexo, por CPO: $\frac{\partial L}{\partial w} = w^T - \sum_{i=1}^{N} \alpha_i y_y x_i^T = 0$
$\implies \bar{w} = \sum_{i=1}^{N} \alpha_i y_i x_i$ y $\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_y = 0$
$\implies \sum_{i=1}^{N} \alpha_i y_i = 0$ por lo tanto se tiene que $\theta(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$. Así, el problema dual resulta:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.a.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned} \tag{3}$$

Para el caso de margen suave, se permiten puntos mal clasificados (cosa que no aceptaba el método anterior de margen duro), es por esto que se le agrega una variable más al problema de optimización, sin embargo, este resulta en la misma estructura y tipo de resolución con algunas restricciones sobre las variables de holgura. A continuación se escribe el problema de margen suave:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}||w||^2 + c \sum_{i=1}^{N} \xi_i \\ \text{s.a.} \quad & y_i(w^T x + b) \geq 1 - \xi_i, \ i \in 1, ..., N \\ & \xi_i \geq 0 \ i \in \{1, ..., N\} \end{aligned} \tag{4}$$

Donde c es un hiperparámetro del problema y ξ_i son las variables de holgura. Notar que si $\xi_i = 0$ el dato x_i está correctamente clasificado y se obtiene el problema anterior. Si $0 < x_i < 1$, el dato x_i está en el lado correcto del hiperplano, pero dentro del margen. Finalmente, si $\xi_i > 1$, el punto x_i está en el lado incorrecto del hiperplano. Luego, de forma similar, el problema dual viene dado por:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.a.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq c \end{aligned} \tag{5}$$

Donde este problema también es de tipo programación cuadrática y su solución es similar al dual de margen duro (existe y es única).

(b) Sin pérdida de generalidad, se puede pensar en el círculo interno centrado en el origen, así, para este caso, un posible kernel (transformación), es uno polinomial, es decir, pasamos de un conjunto de datos en $\mathbb{R}^2$ a $\mathbb{R}^3$ mediante $\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$. Así, las coordenadas x_1, x_2 formaran una parábola cada una (paraboloide en $\mathbb{R}^3$), mientras que la tercera dimensión dada por ϕ genera el cúmulo de puntos relacionados al círculo. Por otro lado, un posible kernel es también el rbf, pues es uno que, en infinitas dimensiones es posible separar el conjunto mediante un hiperplano, donde actúa bien con datasets radiales debido a la forma gaussiana que tiene, $K(x, y) = exp(-\frac{||x-y||^2}{2\sigma^2})$. En la siguiente imagen se ve un ejemplo del kernel polinomial a modo ilustrativo:

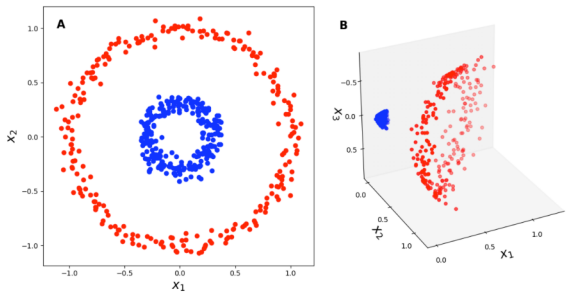

Figura 1: Separación del conjunto en $\mathbb{R}^2$ a $\mathbb{R}^3$ por un kernel polinomial.

(c) Notemos que el conjunto no es linealmente separable por definición, ya que cualquier hiperplano tendrá ambas clases presentes en cada lado. Luego, al igual que en la parte (b), se hace el supuesto que los datos están centrados en el origen. Así, si definimos el kernel polinomial mediante $K(x_i, x_j) = (\langle x_i, x_j \rangle + c)^d$, podemos elegir los parámetros $c = 1$, $d = 2$ y obtener $K(x_i, x_j) = (\langle x_i, x_j \rangle + 1)^2 = (x_i x_j^T + 1)^2$ ya que estamos trabajando en el espacio euclideano. Usando esta transformación a los datos, resultará en un espacio de características $[x_i^2, x_j^2, \sqrt{2}x_i x_j, 1]$. Notemos que esto forma una especie de paraboloide en $\mathbb{R}^3$, donde sólo falta comprobar que los datos de ambas clases puden ser separados mediante un hiperplano. Dada la naturaleza del anillo exterior de puntos, si una coordenada, por ejemplo, x_1 aumenta en magnitud ($||x_1||$), la otra coordenada debe acercarse a 0, esto pues sino, no se tendría una distribución tipo anillo. Supongamos el radio del anillo igual a R. Acá, si una coordenada es 0, la otra debiese ser R o $-R$. Por otro lado, la otra clase es un círculo centrado en el origen, donde ambas coordenadas no pueden ser mucho más grandes que 0, digamos r. Así, calculando las nuevas características de la clase del círculo dadas por el kernel resulta que, como x_1, x_2 son cercanos a 0 ($< r$), entonces $\sqrt{2}x_1x_2 \sim 0$. Por otro lado, en la clase del anillo, si alguna coordenada es cercana a 0 (por ej, x_1), la otra debe ser R o $-R$ por lo cual el conjunto de características tiene el valor R^2 en la segunda coordenada, alejándose completamente de la otra clase, y formando un posible hiperplano de separación si tomamos en cuenta todos los puntos del conjunto.

(d) Para esta parte, notemos que cambiar valores de d, sólo afecta al grado de los polinomios obtenidos como nuevos *features*. Así, dado un $d \in \mathbb{N}$, las nuevas características estarán formadas por todos los monomios de grado $\leq d$ junto con el término libre c si $c \neq 0$. Es decir, que aumentando el valore de d, solo estamos obteniendo aún más información, por lo cual como se vió en la parte (c), para cualquier $d \geq 2$, el conjunto será linealmente separable en ese nuevo espacio (de más dimensiones). Notar también que si $c = 0$ y $d \to \infty$, el kernel polinomial se empezará comportar como el rbf (sin llegar a ser como este pues no podemos seleccionar $d = \infty$), donde sabemos que este último kernel, en espacio infinito-dimensional, es capaz de separar mediante un hiperplano cualquier conjunto inicial.

P2: Pregunta práctica

1. Árboles de Decisión. (a) La poda de un árbol consiste en elegir un nodo y el subárbol que posee este nodo elegido como raíz y extraerlo del conjunto (árbol) entero. Esto se hace con el objetivo de evitar el sobreajuste y disminuir la complejidad del árbol entero. Un criterio posible para esta poda es el de **minimal cost complexity**, y consiste en, dado una métrica $R(T)$ que represente el costo de un árbol T, por ejemplo, la tasa de error de clasificación, se define una nueva métrica dada por: $R_\alpha(T) = R(T) + \alpha|T|$, con $|T|$ número de hojas totales. Así, el parámetro α es el encargado de penalizar más o menos la complejidad del árbol como costo en complejidad de un nodo terminal. Luego, la idea del algoritmo de poda con criterio **minimal cost complexity** consiste en construir una sucesión de subárboles $T_0, T_1, ..., T_m$, con T_0 el árbol original, tal que en cada paso haya menos nodos terminales y T_m representa la raíz. El pseudo código del algoritmo puede verse en el Anexo o en el ipynb. Finalmente, lo que obtenemos de esto es una sucesión de subárboles y de α's. Así, para elegir con qué quedarse, una manera razonable de escoger un árbol final es tomar aquel que tenga

menos error en un conjunto de entrenamiento o validación. También podemos hacer uso de validación cruzada.

(b) Luego de crear y entrenar los árboles, se obtiene un *accuracy* de 1.0 y 0.878 en los conjuntos de entrenamiento y prueba respectivamente. Notar que el *accuracy* en el conjunto de entrenamiento es 1, pues por defecto, estamos haciendo que cada punto sea clasificado de manera correcta, obteniendo así un 100 % de predicción. En el gráfico anterior se puede ver que la profundidad máxima del árbol es 13. Esto recordando que la raíz (primer nodo) es de profundidad 0, por lo cual se empiezan a contar desde los hijos de este nodo.

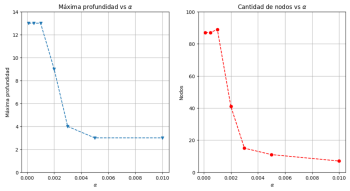

Figura 2: Gráfico max_depth y cantidad de nodos vs α.

En estos gráficos se aprecia que tanto la profundidad máxima del árbol como la cantidad de nodos va disminuyendo en cuanto α va incrementando. Esto tiene sentido pues α actúa como un parámetro de penalización, por lo cual el árbol empieza a quedarse más pequeño.

(c) Dado el gráfico mostrado, se ve que el accuracy en el conjunto de entrenamiento va disminuyendo, sin embargo, se puede observar que en el conjunto de prueba, el accuracy va subiendo, es decir, que mientras α va creciendo, la robustez del modelo y su capacidad de generalización también aumenta por lo que nos quedaremos con un $\alpha = 0.01$ en este caso. Notar que se probaron los valores $\alpha \in [0.001, 0.01]$. Aumentar demasiado este hiperparámetro del modelo puede hacerlo ineficiente podándolo en exceso.

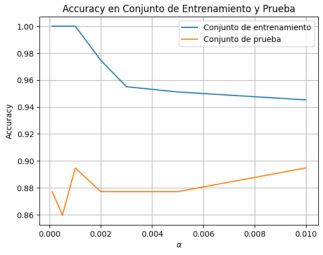

Figura 3: Accuracy en DecisionTree

Acá se puede observar como el árbol es bastante más compacto considerando la poda con el criterio de **minimal cost complexity** y un $\alpha = 0.01$. Comparando este árbol con el de parámetros predefinidos, se puede ver que el accuracy en el conjunto de prueba es prácticamente idéntico, sin embargo, en el conjunto de entrenamiento, la mayor profundidad del primero modelo hace que este sea mayor que el modelo con $\alpha = 0.01$, sin embargo, podemos concluir que este score no afecta la generalización de los 2 modelos pues poseen mismo score en conjunto de prueba. Más aún, al ser el árbol podado más compacto, existe una ganancia en la fácil interpretabilidad que posee el clasificador. Es por estas dos razones que, para decidir un modelo a utilizar, se prefiere el árbol podado con $\alpha = 0.01$. Este árbol se puede ver en el Anexo o en el ipynb.

(d) La principal diferencia entre un modelo de **Bagging de árboles** y **Random Forest** que es ente último, no se seleccionan todas las características del conjunto entregado, es decir, se seleccionan $m < M$ donde M es el número total de *features*. Esta selección se hace de forma aleatoria, mientras que el método de bagging usa todas las M variables.

En conceptos más fundamentales, esta diferencia viene pues la varianza del algoritmo de bagging con árboles es $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$ donde ρ es la correlación 2 a 2 de los estimadores, σ^2 es la varianza de cada estimador (pues son iid) y B es el número de árboles. Así, RF quiere disminuir el término $\rho\sigma^2$ reduciendo la correlación de los estimadores para reducir la varianza sin afectar fuertemente el sesgo.

(e) Como es sabido, el algoritmo de random forest escoge un subconjunto de variables que considera más importantes en relación a la variable target. Existen distintas formas de calcular esta importancia de variables. Una posible opción es el **Criterio de Gini**, el cual es un valor en el intervalo $[0, 0.5]$ e indica informalmente la probabilidad de que un nuevo dato aleatorio sea mal clasificado si ha sido dotado de una clase al azar de acuerdo a la distribución de clases del dataset. Matemáticamente, si se tiene un conjunto de datos D con datos de k clases, denotamos p_i la probabilidad de que un dato pertenezca a la clase i. Así, la impureza del dataset es $Gini(D) = 1 - \sum_{i=1}^{N} p_i^2$. Luego, si el dataset se ha dividido por un atributo A en los conjuntos D_1, D_2 de tamaño n_1, n_2 respectivamente, entonces la impureza de Gini resulta: $Gini_A(D) = \frac{n_1}{n}Gini(D_1) + \frac{n_2}{n}Gini(D_2)$, donde $n = n_1 + n_2$. Con estas definiciones, se escoge el atributo con menor impureza de Gini. En caso de tener varios árboles, se pueden promediar estos valores para obtener una ïmportancia final".

Por otro lado, existen otros criterios de importancia que no se detallarán por temas de espacio, tales como el criterio de **entropía** y **log loss** que están relacionados con la teoría de la información

(Shannon).

2. Support vector machines. (a) En la formulación de SVM con margen suave, se puede entender al hiperparámetro c tal que si $c \to \infty$, entonces se recupera el problema de máximo margen duro con su misma solución en caso de que el conjunto de datos sea linealmente separable. Por el contrario, si c va tomando valores más pequeños, se le resta importancia a los datos mal clasificados, obteniendo un margen más amplio. Es por esto que aumenta $||w||$ al aumentar C.

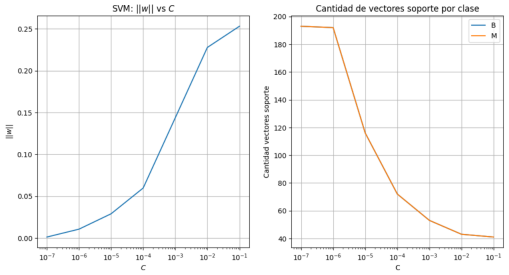

Figura 4: Parámetros de SVM vs C.

Con respecto a los gráficos, se puede observar que, mientras aumenta el valor de C, la norma del parámetro también aumenta, sin embargo, la cantidad de vectores de soporte va disminuyendo. Esto se deba a que, como se está recuperando el problema de margen duro, sólo unos pocos vectores serán de soporte debido a que es difícil que más vectores tengan exactamente la misma distancia al hiperplano separador. Por temas del dataset, en este gráfico de la derecha, la cantidad de vectores soporte por cada clase (B, M) es idéntica por lo que se traslapan sus curvas.

(b)

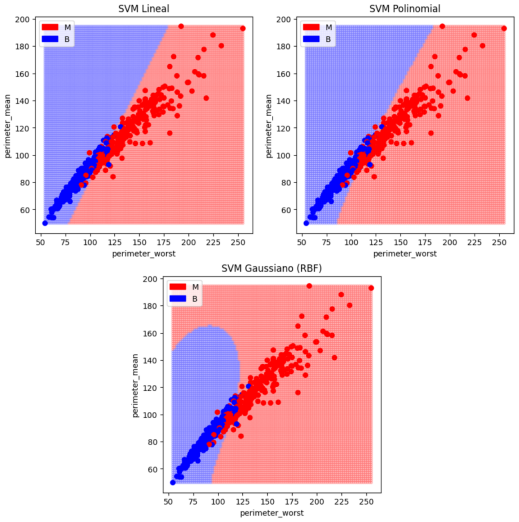

Figura 5: Regiones de decisión SVM lineal, polinomial y gaussiano.

Acá se pueden apreciar los gráficos que resultan de aplicar el modelo SVM para clasificar los datos. En estos se pueden ver los datos reales en forma de puntos, mientras que, de tonalidad más clara, se pueden ver las fronteras de decisiones que se obtienen de SVM al usar distintos kernels. Se puede ver que, para los parámetros por defecto de sklearn, el kernel lineal y polinomial arrojan el mismo resultado, sin embargo, el kernel gaussiano genera una región completamente distinta para su clasificación. Por último, debido a que los datos no se ven en gran medida separables linealmente, se considera el uso de SVM con kernel gaussiano una mejor opción para este problema.

3. Selección de modelo. (a) En el archivo ipynb se crea una función que permite calcular la curva ROC. Esta curva es tal que, grafica los ratios de verdaderos positivos en el eje y (predicciones correctas del modelo a una clase) y el ratio de falsos positivos en el eje x (datos que el modelo predijo la otra clase incorrectamente). Así, mientras la curva ROC esté más alejada de la función identidad $f(x) = x$, indica un mejor resultado de las probabilidades dadas por un modelo de clasificación, mientras que una curva cercana a a identidad, indica un mal desempeño del clasificador. Notar que la línea que define la función identidad representa a un clasificador aleatorio con probabilidad $1/2$ a cada clase.

(b)

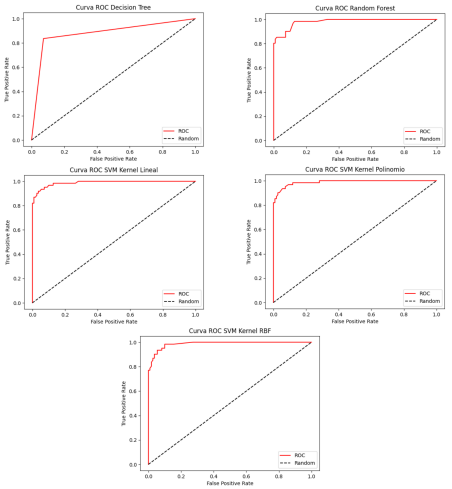

Figura 6: Curvas ROC para cada modelo.

Como se puede ver, se ha logrado satisfactoriamente un método para poder graficar la curva ROC en estos clasificadores. Notar que, como se dijo en la sección anterior, mientras la curva más se acerque al punto $(x, y) = (0, 1)$, más se acerca el clasificador a predecir todo correctamente. Así, se puede observar como todos los clasificadores poseen curvas similares, a excepción del modelo más simple puesto a prueba, el árbol de decisión. Esto sucede pues es un método menos robusto para este

problema que los otros, y está construido con los parámetros predeterminados de sklearn. Además, su construcción no está basada en darle probabilidades a cada clase por lo que su gráfico tiene esta forma más simple.

(c) Observando los gráficos de la sección (b), para este problema parece adecuado elegir el modelo de clasificación SVM con kernel lineal. Esto ya que en el gráfico de las curvas ROC, parecen estar correctamente predichas las clases en más datos que en los clasificadores DecisionTree y RandomForest. Comparando luego los distintos kernels que posee el algoritmo de SVM, estos no parecen agregar demasiadas mejoras al modelo, por lo cual guiándose por el tipo de complejidad computacional que tienen estos algoritmos, el modelo lineal es más rápido y ocupa menos memoria que el kernel polinomio y rbf. Esto claramente depende de los datos a trabajar, sin embargo, con este dataset parece ser una buena opción a utilizar. Ventajas de utilizar la curva ROC son las siguientes:

- Fácil implementación y rápida ejecución.
- Permite encontrar el umbral óptimo según el problema
- Interpretable bajo los conceptos de verdaderos y falsos positivos.

Desventajas de utilizar la curva ROC son las siguientes:

- No tiene en cuenta los costos asociados con los errores de clasificación.
- No distingue entre diferentes tipos de errores (Tipo I y Tipo II).
- Puede no comportarse bien con datos demasiado desbalanceados.

Bajo el contexto de este dataset con información de tumores cancerígenos benignos o malignos, el objetivo es tener la menor cantidad posible de falsos negativos, esto pues en medicina es más importante detectar todos los pacientes con cáncer maligno que equivocarse habiendo etiquetado malignos a un paciente con un cáncer benigno. Es por lo anterior que la métrica que más se quiere cerca de 1 es el **recall**. Para esto, se puede ver directamente en las matrices de confusión, o utilizar alguna métrica que la considere, como por ejemplo **f1-score**. En el Anexo o en el archivo ipynb se pueden ver las métricas obtenidas por cada modelo. Con esto, finalmente se verifica de mejor manera que el modelo óptimo para este problema es el SVM con kernel lineal, pues posee mayor recall para la clase 1 (Maligno) obteniendo también la mejor puntuación en la métrica f1-score.

Anexo

1. Árboles de decisión y Random Forest (a)

```
# Función de poda al árbol T
def Poda de costo-complejidad():

  #Función de generación de sucesión de subárboles
  def SucesionArboles(T):
    # Constantes
    k = 0
    T_0 = T
    alpha = np.inf # infinito

    for t in nodos no terminales desde abajo hacia arriba:
      r = R(T_t)
      t = |T_t|
      g = (R(T) - R(T_t)) / (|T_t| - |T|)
      alpha = min(alpha, g)

    for t in nodos no terminales de arriba hacia abajo:
      if g = alpha:
        reemplazar T_t por t # Podar
        k = k + 1
        alpha_k = alpha
        T_k = T

    if T es un árbol trivial:
      return alpha_1,..., alpha_m, T1, ..., T_m
```

Figura 7: Pseudo código de poda de árboles.

1. Árboles de decisión y Random Forest (c)

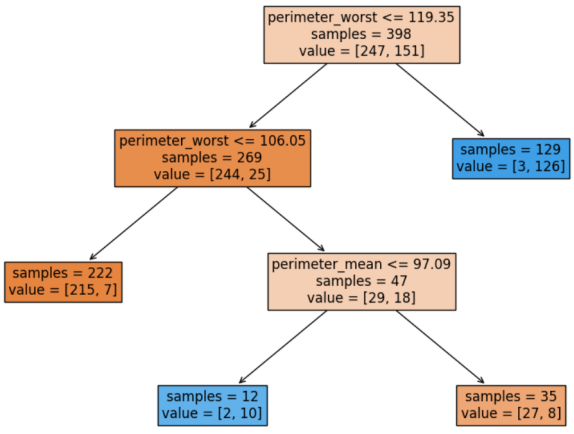

Figura 8: Árbol de decisión con $\alpha = 0.01$.

3. Selección de modelo (c)

Árbol de Decisión:

	precision	recall	f1-score	support
0	0.91	0.93	0.92	110
1	0.86	0.84	0.85	61
accuracy			0.89	171
macro avg	0.89	0.88	0.88	171
weighted avg	0.89	0.89	0.89	171

Random Forest:

	precision	recall	f1-score	support
0	0.92	0.97	0.95	110
1	0.95	0.85	0.90	61
accuracy			0.93	171
macro avg	0.93	0.91	0.92	171
weighted avg	0.93	0.93	0.93	171

Figura 9: Métricas de DecisionTree y RandomForest.

SVM Lineal:

	precision	recall	f1-score	support
0	0.93	0.98	0.96	110
1	0.96	0.87	0.91	61
accuracy			0.94	171
macro avg	0.95	0.93	0.93	171
weighted avg	0.94	0.94	0.94	171

SVM Polinomial:

	precision	recall	f1-score	support
0	0.92	0.98	0.95	110
1	0.96	0.85	0.90	61
accuracy			0.94	171
macro avg	0.94	0.92	0.93	171
weighted avg	0.94	0.94	0.93	171

SVM RBF:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	110
1	0.96	0.80	0.88	61
accuracy			0.92	171
macro avg	0.93	0.89	0.91	171
weighted avg	0.92	0.92	0.92	171

Figura 10: Métricas de SVM con distintos kernels.