Interpretable Machine Learning

Iñaki Inza, Borja Calvo

La búsqueda de la "transparencia en la intepretabilidad" de los modelos aprendidos y la "explicabilidad de las predicciones" realizadas es un hot-topic en el mundo del machine learning. Quizás debido a la disruptiva entrada de los modelos deep-learning: con una gran capacidad para resolver problemas complejos en entornos de imágenes, texto, voz, etc... pero con un marcado cariz "black-box caja-negra" que no hace posible ni interpretar su estructura ni entender sus predicciones. Creo que su irrupción ha hecho que parte de la comunidad "tirará" por otro lado, investigando en herramientas que nos abrieran estas posibilidades de "interpretability".

Dentro del "boom" de papers, conferencias, libros, manuales, etc... que ya se pueden encontrar al respecto, con el objetivo de primero yo aprender, y posteriormente trasladaros un resumen de lo aprendido, me he basado principalmente en un fantástico libro on-line [2]. Como indica en su título, "a guide for making black box models explainable". Recojo de éste una serie de herramientas en este camino de la "intepretability", siendo cierto que me dejo por el camino unas cuántas que creo nos quedan lejos, o sin más, no las entiendo lo suficiente como para poder resumiroslas.

El conocimiento que recoge el libro se complementa con el paquete `iml - interpretable machine learning`, y que nos resume rdocumentation sus funciones; así como el repositorio GitHub del autor en el que se basan libro y paquete.

> Se ejemplificarán todas estas medidas para un escenario de clasificación donde la variable output es nominal (i.e. clasificación supervisada). Su extensión para problemas de clasificación donde la variable output es numérica (i.e. regresión) está disponible también en múltiples manuales en la red. La interpretación de todas estas medidas para el caso de problemas de regresión es quizás más intuitiva.

Año:
Profesor(a): Iñaki Inza, Borja Calvo
Teléfono: 943 01 50 26
E-mail: inaki.inza,borja.calvo@ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

1 Feature Importance - peso de cada feature en el performance del modelo

La función `featureImp()` nos permite estimar, dado un modelo predictivo ya aprendido, la importancia de cada variable predictora respecto a las predicciones que realiza el modelo. Resumiendo mucho, ¿cómo se calcula esta "importancia"? En el dataFrame que recoge el dataset de entrenamiento, se reordena aleatoriamente la columna de la variable de interés, se vuelve a aprender un modelo, y la caída en performance de éste se asimila a la importancia de la variable en dicho modelo: "permutation feature importance". Este proceso se realiza varias veces y se muestran los resultados agregados.

Para extender la información sobre este cálculo, detalles, etc., consultar la sección sobre "permutation feature importance" del libro del autor del software.

```
# Feature Importance -- peso de cada feature en el performance del modelo
library(iml)
library(caret) #instálalas previamente si es que no lo estaban
data("iris") # problema aburrido pero pequeño para poder analizarlo
summary(iris)
TrainData <- iris[,1:4]
TrainClasses <- iris[,5]
# aprendemos un modelo predictivo;
# que puede ser uno aprendido con la librería "caret";
# modelo "naive_bayes" se implementa en el paquete "naivebayes";
# listado de algoritmos de clasificación ofrecidos por caret:
# https://topepo.github.io/caret/train-models-by-tag.html
nbFit <- train(TrainData, TrainClasses, "naive_bayes", tuneLength = 10,
               trcontrol=trainControl(method="cv"))
# creamos mediante "iml" un objeto que incluye el modelo aprendido para su análisis
model <- Predictor$new(nbFit, data=TrainData, y= TrainClasses)
# métrica de evaluación -> "cross-entropy" de la clase real
imp <- FeatureImp$new(model, loss="ce")
# interpreta gráfica y compara resultados numéricos entre features en forma de "proporciones";
# no como valores absolutos
plot(imp)
imp$results
```

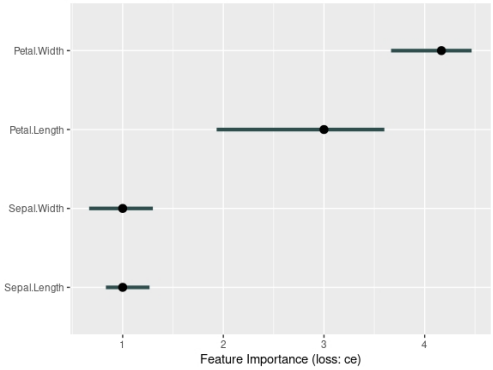

Figura 1: 'Feature importances' de las 4 predictoras en el problema del 'iris prediction', sobre un modelo naive Bayes. La métrica de evaluación ha sido 'cross-entropy' sobre la clase real.

2 Feature effects - cómo influyen las predictoras en la predicción

Damos un salto, y ahora nos interesa estimar la 'influencia de cada predictora en la predicción del modelo'. Hay varias propuestas en la literatura, entre ellas la 'Accumulated Local Effects - ALE'. Calculará cómo la clase predicha por el modelo cambia localmente, alrededor de cada valor concreto de nuestra variable predictora: variando el valor de la predictora, cuál es el cambio en la clase predicha.

Estamos interesados en calcular el valor ALE para la variable X_i. Para ello, necesitamos un clasificador con output probabilista (e.g. red Bayesiana, neural network, etc.). Para cada caso, se calcula cómo varía la probabilidad de pertenencia a cada clase, $p(clase|(x_1, x_2, ..., x_i, ..., x_d))$ al variar localmente los valores en una ventana alrededor del valor x_i: y manteniendo fijos el resto de valores de la instancia.

La predictora X_i es dividida en intervalos. Para los casos en cada intervalo y tal y como se ha indicado previamente, se calcula la *diferencia* en la mencionada probabilidad de pertenencia a cada clase cuando se reemplaza el valor real de la predictora por el del 'upper' y 'lower' límites del intervalo. Estas diferencias en la probabilidad de pertenencia predicha se aculuman y se muestran en una intuitiva gráfica. Figura 2.

Para extender la información sobre este cálculo, detalles, etc., consultar la sección sobre "Accumulated Local Effects - ALE" del libro del autor del software.

```
# ALE para PetalLength sobre el modelo previamente aprendido
ale <- FeatureEffect$new(model, feature="Petal.Length")
ale$plot()
# Para todas las predictoras
effs <- FeatureEffects$new(model)
# "patchwork" library is needed for plotting
plot(effs)
```

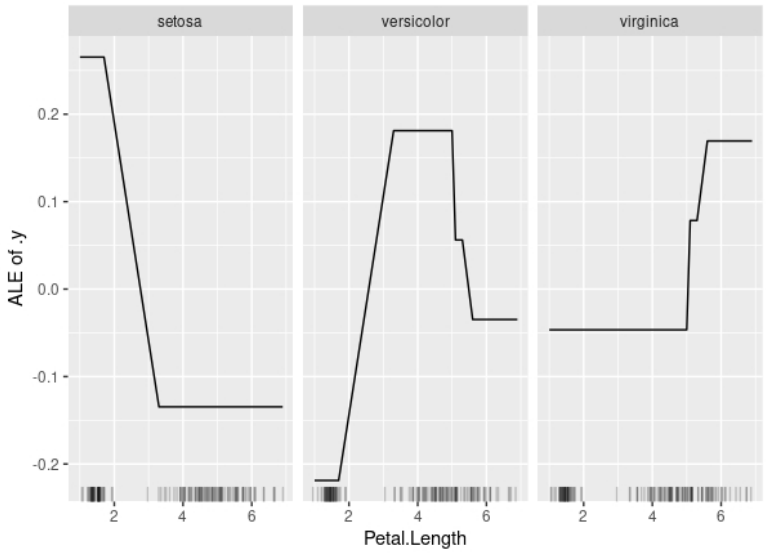

Figura 2: 'ALE effect' de la predictora 'Petal Length' en el problema de 'iris prediction'. Una visualización por cada clase: cada una se calcula a partir de un modelo aprendido en formato "one-class versus rest-of-the-classes". Las líneas verticales del eje horizontal muestran la densidad de valores de la variable: por ello, el valor ALE en las zonas de baja densidad no debe tomarse en cuenta.

3 Feature interactions - cómo interaccionan las predictoras entre sí para inferir la predicción?

Cuando las predictoras interactúan en el modelo predictivo, la predicción no puede expresarse como la suma de efectos individuales: el efecto de una puede depender en el valor de otra. De forma simplificada, en un modelo predictivo compuesto por dos predictoras, la predicción puede descomponerse en 4 términos: una constante, un término para la primera predictora, uno para la segundo, y un término para la interacción entre ambas. Este último término recoge el cambio que ocurre en la predicción al variar las predictoras, pero tras considerar los efectos individuales. Si tras considerar los efectos individuales sigue habiendo varianza en la predicción, es que la '2-way interaction' no es nula. El Friedman's H-statistic nos ofrece una fórmula para calcular esta '2-way interaction'. Ejemplo para la variables 'Petal.Length' respecto al resto de predictoras, en nuestro modelo previamente aprendido (Figura 3).

```
# 2-way interaction entre Petal.Length y el resto predictoras;
# se realiza para cada problema de clasificación `one-class versus rest-of-the-classes'
interact <- Interaction$new(model, feature="Petal.Length")
plot(interact)
```

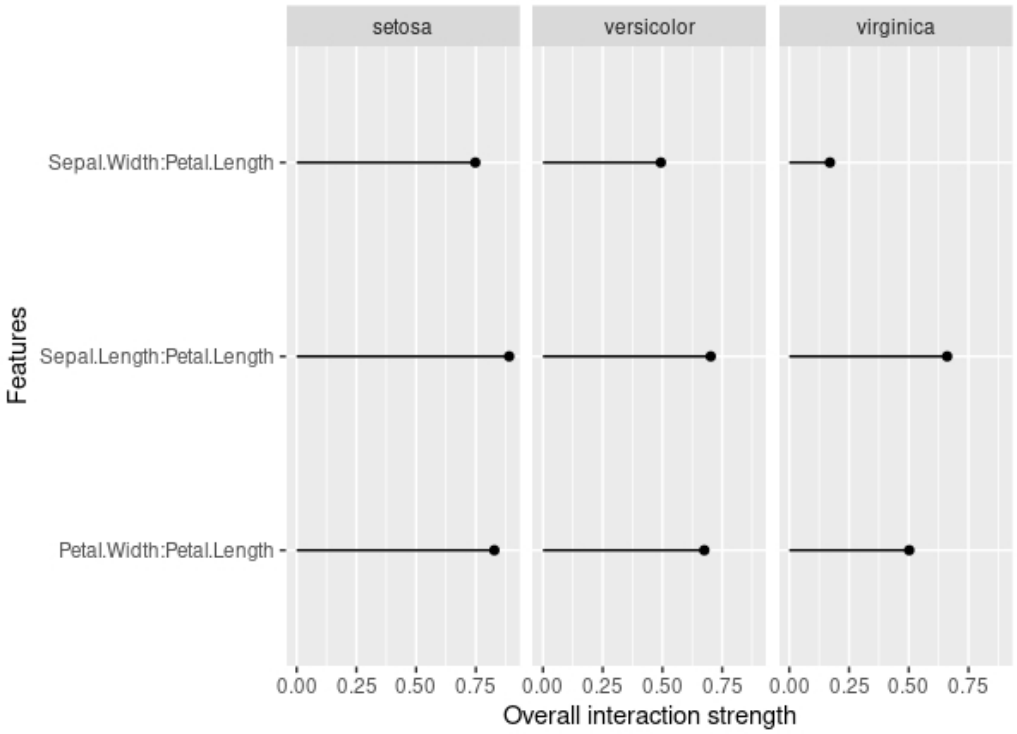

Figura 3: '2-way interaction' de la predictora 'Petal Length' en el problema de 'iris prediction'. Una visualización por cada clase: cada una se calcula a partir de un modelo aprendido en formato "one-class versus rest-of-the-classes".

4 Surrogate trees

Cuando nos encontramos con un clásico modelo 'black-box' ('performance' convincente pero ausencia de transparencia y comprensión del modelo), esta idea de los 'surrogate trees' es sencilla y a la vez elegante. Se propone crear un dataset con las predicciones realizadas por el modelo black-box para dichas muestras. Tomando estas predicciones como variable-clase y manteniendo las predictoras del problema original, se propone aprender un modelo de árbol de clasificación sobre las predictoras originales y la clase predicha por el modelo 'black box'.

Un árbol de clasificación, con su intuitiva estructura, nos ofrece una explicación de 'cómo clasifica nuestra modelo black box'. Ojo, no si lo hace correctamente o incorrectamente. Sino cuales son sus reglas (en formato de árbol de clasificación) para generar las predicciones que propone. 'Darle explicabilidad', mediante este 'surrogate tree', a sus predicciones: ya que el modelo no es transparente. Ejemplo en Figura 4.

```
# aprendemos un multi-layer perceptron
mlpFit <- caret::train(TrainData, TrainClasses, "mlpML", tuneLength = 2)
model <- Predictor$new(mlpFit, data=TrainData, y= TrainClasses)
# aprendemos el surrogateTree que trata de explicar cómo clasifica el MLP
surrogateTree <- TreeSurrogate$new(model, maxdepth = 2)
plot(surrogateTree)
```

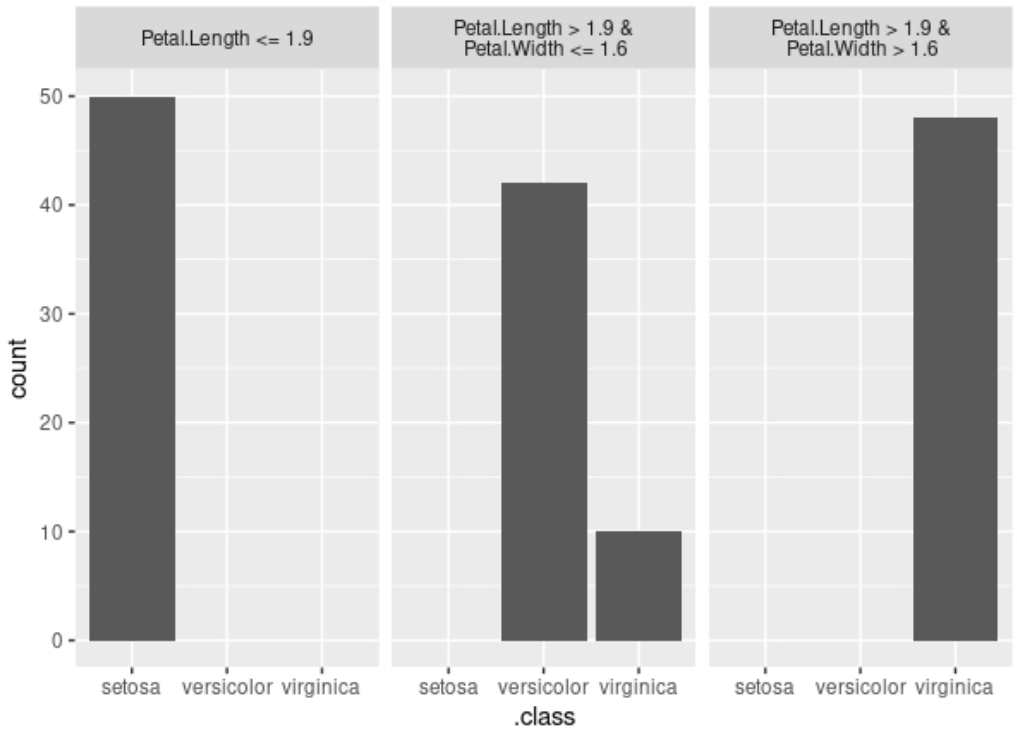

Figura 4: SurrogateTree para ayudarnos a explicar cómo clasifica el multi-layer-perceptron, cuáles son sus reglas de predicción, en formato de árbol.

5 SHAP values: midiendo el impacto de cada variable en la predicción final

Dado un modelo predictivo ya aprendido y un caso a ser predicho su output, es interesante conocer el *impacto* de cada variable (predictora) del modelo en la predicción de ese caso. Ésto viene a englobarse en el llamado 'interpretable machine learning', centrado en entender porqué los modelos predicen lo que predicen.

Este problema está levantando vivamente la atención de muchos campos de aplicación. Uno es la comunidad biomédica. El interés en conocer qué parámetros-variables 'han hecho' que la predicción del diagnóstico-pronóstico del paciente sea el recibido. Pero

Estos valores del impacto de cada variable son conocidos como 'SHAP value', en honor a su inventor Lloyd Shapley, que los propuso en un entorno de teoría de juegos [3]; y han sido adaptados para formular el problema expuesto. La idea de Shapley era que en un 'juego de coalición', es conocido el beneficio conseguido por un equipo. Siendo el objetivo repartir entre los jugadores de una manera justa el beneficio conseguido, para ello se debe tener en cuenta cada 'sub-coalición sub-equipo', valorando cuánto le aporta la presencia de cada jugador a cada 'sub-coalición sub-equipo'. Los jugadores vendrían a ser las variables predictoras. Cada 'sub-coalición sub-equipo', un subconjunto de variables. Posteriormente lo entenderemos mejor. Vayamos a un ejemplo.

En la Figura 5 se muestran, en formato nube de puntos (i.e. densidades), los valores de SHAP para cada valor de cada variable. Cada punto viene a representar un caso, con su valor correspondiente en cada variable-predictora: variables, en el eje vertical izquierdo, alineadas. El valor de SHAP, en el eje horizontal. Recordemos que este cálculo se hace para casos cuyo output debe ser predicho por el modelo 'ya' aprendido.

Fijémonos en la 'Feature 6': existen múltiples casos con valores bajos de esta variable (color azul, eje vertical derecho), que 'empujan' hacia un valor positivo-superior en el output de la clase. Y múltiples casos con valores altos (color rojo) 'empujan' hacia una predicción negativa-inferior de la clase, incluso con una 'fuerza mayor', hacia un valor más extremo-negativo del output. Sigue tú mismo con otra variable e interpreta...

Para interpretar el output de la clase, lo más intuitivo es pensar que bien estamos ante un problema de regresión-numérica, o bien ante un problema de clasificación ordinal donde los dos valores nominales de la variable clase tienen un orden (e.g. pronóstico fallecimiento, pronóstico supervivencia).
La semántica de estos modelos es muy potente, y cada vez son más populares en aplicaciones de distinto tipo donde hay interés en conocer qué variables han hecho que un caso a predecir reciba un output concreto.

¿Cómo se calculan estos valores? No es trivial, me ha llevado un buen rato avanzar en su comprensión; trataré de ofreceros la intuición del procedimiento [4, 1].

En los modelos aditivos, e.g. $f(x_1, x_2) = 2x_1 - 3x_2 + 4$, la contribución del valor de cada variable $X_i = x_i$ en la predicción realizada por f es la misma para todas las instancias. Esto es así porque las variables no interactúan entre sí, y el coeficiente de las variables en el modelo nos indica la magnitud de la contribución. Pero en otro tipo de modelos las variables sí interactúan, y la contribución-impacto del valor $X_i = x_i$ de una variable en la predicción depende de los valores del resto de variables. Os propongo un breve ejemplo.

Tenemos un modelo clasificatorio f ya aprendido, y el caso $x = (X_1 = 7, X_2 = 9, X_3 = 8, X_4 = 10)$ a ser clasificado. Y nos preguntamos cuál es el 'impacto', el SHAP-value de $X_2 = 9$ en la predicción resultante, i.e. $SHAP_{X_2=9}$. Para ello, los autores proponen calcular el efecto en f de todas las combinaciones que incluyen a $X_2 = 9$: y restando a cada una de éstas el valor de f cuando $X_2 = 9$ es eliminada. De esta manera observamos el efecto de incluir $X_2 = 9$, en cada subconjunto de variables. Intuitivamente, buscamos: ¿cuánto aporta el hecho de 'disponer' de $X_2 = 9$, a cada subconjunto de variables? *El sumatorio de estas diferencias se considera el SHAP-value, el impacto de* $X_2 = 9$ *en la predicción resultante.* Se deben realizar estos cálculos:
$f_{1,2,3,4} - f_{1,3,4}$,

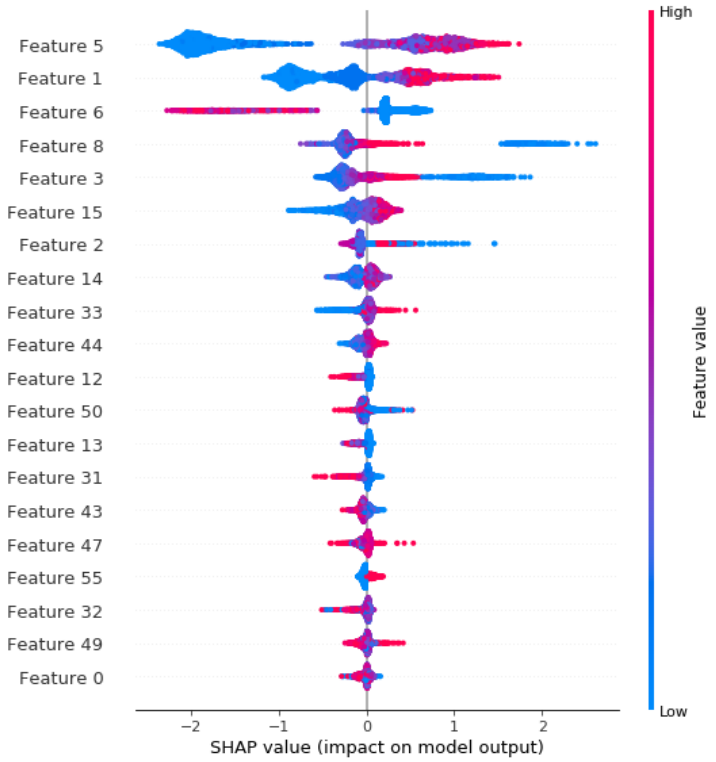

Figura 5: SHAP values. Ejemplo.

$f_{1,2,3} - f_{1,3}$, $f_{1,2,4} - f_{1,4}$, $f_{2,3,4} - f_{3,4}$,
$f_{1,2} - f_1$, $f_{2,3} - f_3$, $f_{2,4} - f_4$
donde, por ejemplo, $f_{2,3,4}$ denota el valor de la función cuando $(X_2 = 9, X_3 = 8, X_4 = 10)$. En un ejemplo con 4 variables es posible realizar todos los cálculos necesarios. Pero a poco que crezca la dimensión, el problema es NP-hard, y no es computacionalmente posible realizar todos los cálculos. Por ello, para aproximarnos al valor real del SHAP-value, se realiza un muestro aleatorio ('sampling') de subconjuntos de variables: esto es, sólo consideraremos una parte de todos los subconjuntos posibles que contengan a $X2 = 9$. ¿Cuántos subconjuntos se muestrean? Se fija una cantidad, en base a los recursos computacionales disponibles.

Algo similar ocurre con los valores de las variables. Por ejemplo, ¿cuál es el valor de X_1 y X_3 cuando en el cálculo del SHAP-values es necesario el de $f_{2,4}$? Esto es, necesitamos calcular el valor de la función f cuando $X_2 = 9$ y $X_4 = 10$. Pero, ¿qué valores fijamos para X_1 y X_3? Se simulará un valor aleatorio para cada variable ausente del subconjunto.

Ésta no es la única metodología existente para el cálculo de los SHAP-values. Encontrarás implementaciones en librerías, y para distintos tipos de clasificadores. Primero, con la librería `kernelshap`. Nos permite calcular los SHAP values para modelos aprendidos desde distintas librerías, `caret, ml3, tidymodels`. Para un modelo simple de *kNN*. Primero para un conjunto reducido de instancias y para aliviar cómputo, visualización tipo "bee-abeja". Segundo, gráfico tipo "waterfall", mostrando los valores SHAP para una instancia concreta.

```
library(kernelshap)
library(shapviz) #para visualizar los valores de "interpretability"
library(caret)
diabetes <- read.csv("https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv")
```

```
colnames(diabetes)
diabetes$Outcome <- as.factor(diabetes$Outcome)
knnModel <- caret::train(Outcome ~ ., data=diabetes,method="knn",preProc=c("center","scale"))
# cálculo SHAP values para 100 muestras aleatorias, aliviando cómputo

subsample100diabetes <- diabetes[sample(nrow(diabetes),100), ]
knnshap <- kernelshap(knnModel, X=diabetes[,-9], bg_X= subsample100diabetes, type="prob")
knnSHAPViz <- shapviz(knnshap) # para visualizar con la librería "shapviz"
# SHAP values centrados en la "clase 1": prueba a quitar "[[1]]"
beeSHAPknn <- sv_importance(knnSHAPViz, kind="bee")[[1]]
beeSHAPknn

waterfallSHAPknn <- sv_waterfall(knnSHAPViz, row_id=50)[[1]]
waterfallSHAPknn$labels$title = "SHAP waterfall values"
waterfallSHAPknn
```

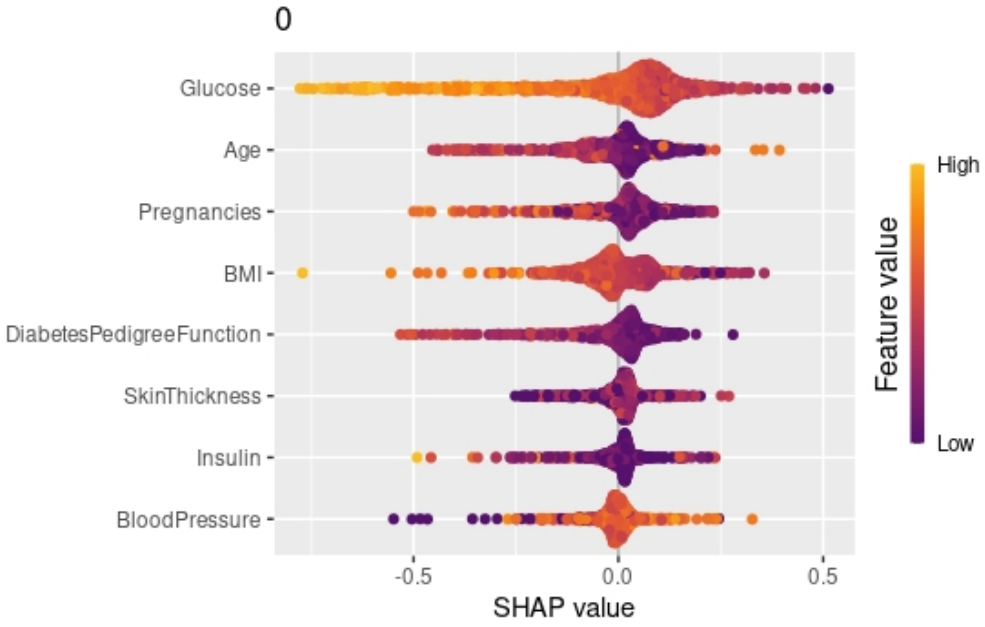

Figura 6: SHAP-values, calculados mediante un modelo knn con la librería "kernelshap", para las predictoras del dataset 'diabetes'

Ahora, con otra librería. Os muestro a continuación una breve demo mediante el paquete de R `SHAPforxgboost`. Los SHAP-values son calculados mediante un clasificador tipo XGBoost, muy popular en las competiciones de kaggle.com. Utilizaremos el popular benchmark `diabetes`, donde el problema de clasificación consiste en el diagnóstico positivo o negativo de la enfermedad para un paciente, en base a unos parámetros-predictores médicos: nos interesan los SHAP-values de éstos, para cada paciente. Encontramos el dataset en múltiples direcciones en la red.

```
library("SHAPforxgboost")
diabetes <- read.csv("https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv")
colnames(diabetes)
X1 = as.matrix(diabetes[,-9]) # dejar las predictoras
# aprender el modelo XGBoost. Fijar predictoras y clase
```

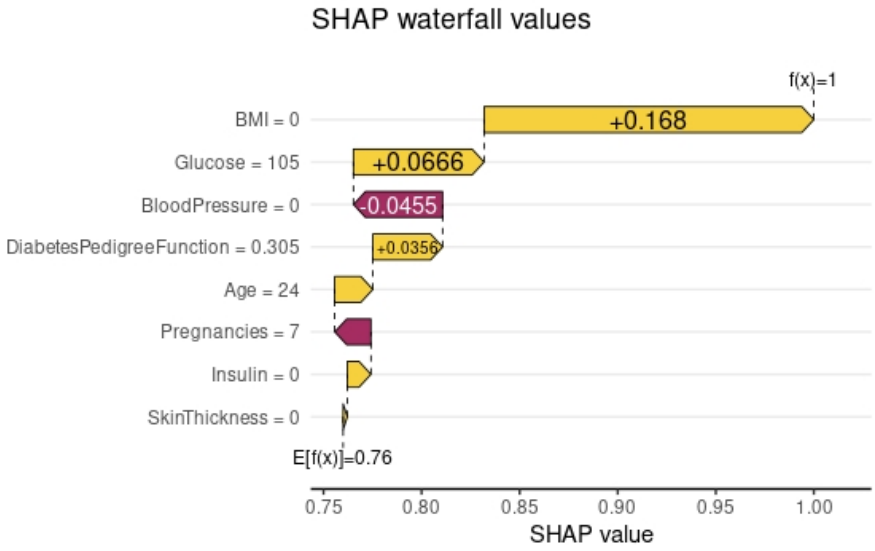

Figura 7: SHAP-values, calculados mediante un modelo knn con la librería "kernelshap", para un paciente de clase negativa. Se observa la aportación de cada variable para finalmente ser predicho como de clase positiva

```
mod1 = xgboost::xgboost(
  data = X1, label = diabetes$Outcome, gamma = 0, eta = 1,
  lambda = 0,nrounds = 1, verbose = FALSE)

# calcular los SHAP values para cada caso - según modelo aprendido
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_diabetes <- shap_values$shap_score
shap_long_diabetes <- shap.prep(xgb_model = mod1, X_train = X1)
# **SHAP summary plot**
shap.plot.summary(shap_long_diabetes)
shap.plot.summary(shap_long_diabetes, x_bound  = 1.5, dilute = 10)
```

Para interpretar la gráfica, recuerda que del origen vertical a la derecha implica una predicción positiva, i.e. diagnóstico positivo de la enfermedad. Fíjate en los valores altos-bajos de las predictoras, cada una en una fila. Recuerda que la gráfica representa una densidad, una nube de puntos, cada uno representando un caso. Por ejemplo, llaman la atención los casos de embarazadas cuyos múltiples partos (color violeta) 'empujan' hacia un diagnóstico positivo (a la derecha de la separación vertical). También algunos pacientes cuyo bajo nivel de presión arterial 'empuja' hacia una predicción positiva de la enfermedad.

Otro ejercicio interesante es individualizar la visualización de los SHAP-values para un caso específico de nuestro interés.

Bibliografía

[1] S.M. Lundberg et al. From local explanations to global understanding with explainable artificial intelligence for trees. *Nature Machine Intelligence*, 2:56–67, 2020.

[2] C. Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. Independently published, 2022.

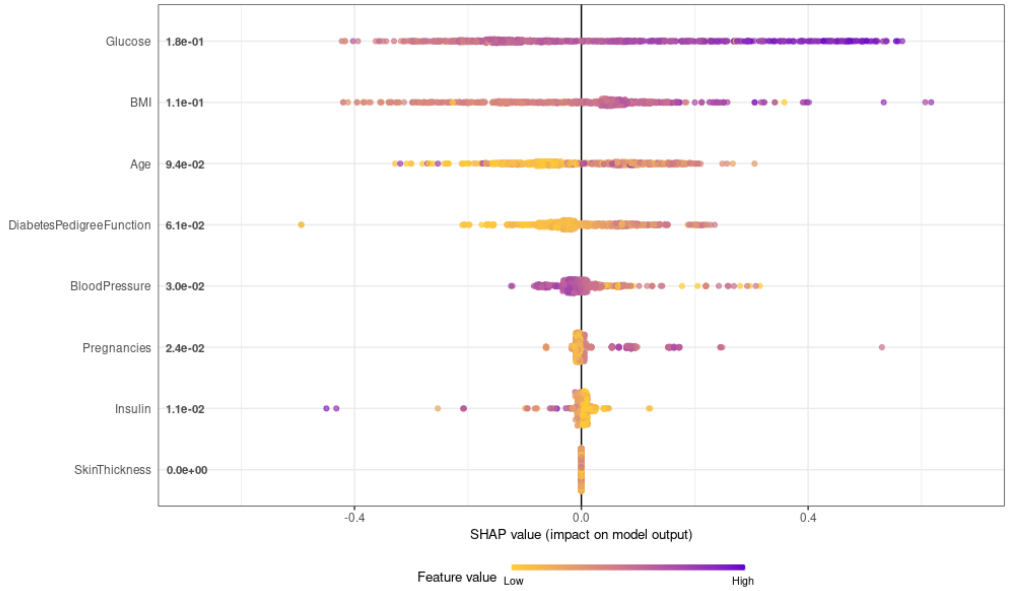

Figura 8: SHAP-values, calculados conn un modelo XGBoost mediante la librería SHAPforxgboost, para las predictoras del dataset 'diabetes'

[3] L.S. Shapley. A value for n-person games. Technical report, Princeton University, 1953. vol II of Contributions to the theory of games.

[4] E. Strumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–665, 2014.