Meta aprendizaje: Boosting y bagging para pronósticos de series temporales

Siempre resulta difícil modelar los cambios en los precios de la gasolina como una variable categórica, especialmente con una pequeña cantidad de datos de series temporales. La solución para mejorar el rendimiento del modelado de un conjunto de datos de este tipo puede ser combinar más de un modelo.

Este método de combinar y agregar las predicciones de múltiples modelos se denomina **meta-aprendizaje** y se basa en un algoritmo que combina modelos más débiles para formar uno más sólido, conocido como **conjunto** (*ensemble*).

Uno de los primeros algoritmos de conjunto que utilizaremos es el **bagging** (agregación bootstrap). El bagging genera numerosos conjuntos de datos de entrenamiento mediante el bootstrap de los datos de entrenamiento originales. Estos conjuntos de datos se utilizan para formar un conjunto de modelos, que cuenta con un único algoritmo, generalmente preferido por los bosques aleatorios, debido a que son **aprendices inestables**, lo que significa que pequeños cambios en la entrada provocan resultados de predicciones significativamente diferentes.

El otro algoritmo de conjunto que utilizaremos es el **boosting**. A diferencia del bagging, este método mejora el rendimiento del modelo al añadirle mejores modelos, lo que significa que forma algoritmos de aprendizaje complementarios. Además, el boosting otorga mayor peso a los algoritmos de aprendizaje en función de su rendimiento pasado, lo que significa que un modelo con mejor rendimiento tiene mayor influencia sobre los conjuntos.

Tras las explicaciones de los algoritmos que utilizaremos, podemos crear nuestro conjunto de datos para los modelos. El conjunto de datos incluye los precios de la gasolina en liras turcas, los precios spot del Brent en dólares estadounidenses y el tipo de cambio USD-TRY entre 2019 y 2022.

Realizaremos algunas modificaciones para ajustar el conjunto de datos a nuestro propósito. Creamos un factor de tres niveles para los cambios en los precios de la gasolina, que será nuestra variable objetivo. Además, añadiremos marcas de tiempo para construir series de tiempo regulares. Para ello, podemos utilizar el paquete tsibble.

```
> library(tidyverse)
> library(tsibble)
> library(readxl)
```

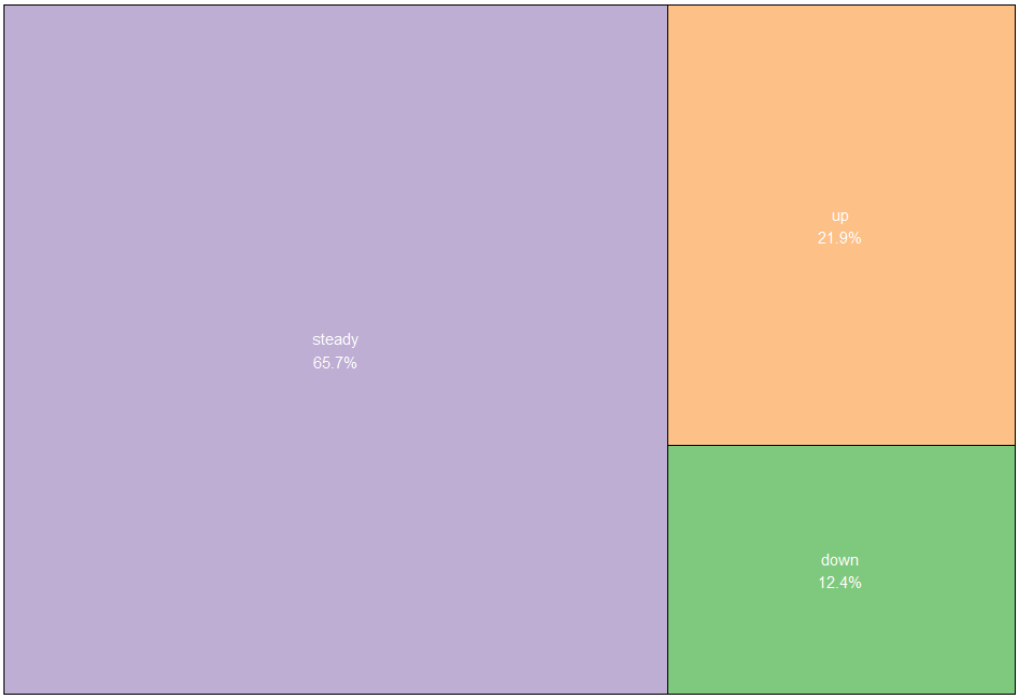
```
> df <- read_excel("gasoline.xlsx")
> str(df)
```

```
tibble [300 × 4] (S3: tbl_df/tbl/data.frame)
 $ date    : POSIXct[1:300], format: "2019-01-01" "2019-01-03" "2019-01-04" "2019-01-08" ...
 $ xe      : num [1:300] 5.29 5.47 5.33 5.48 5.48 ...
 $ gasoline: num [1:300] 4.94 4.94 4.94 4.94 5.08 5.08 5.08 5.21 5.21 5.08 ...
 $ brent   : num [1:300] 53.6 55.1 56.9 58.2 60.9 ...
```

```
> #Building the dataset
> df_tidy <- df %>%
+   mutate(gasoline = case_when(gasoline - lag(gasoline) < 0 ~ "down",
+     gasoline - lag(gasoline) > 0 ~ "up", TRUE ~ "steady") %>% as.factor(),
+   xe_lag = lag(xe),
+   brent_lag = lag(brent),
+   date=as.Date(date)
+   ) %>%
+   as_tsibble() %>%
+   fill_gaps() %>% #makes regular time series by filling the time gaps
+   #fills in NAs with previous values
+   fill(-date,.direction = "down") %>%
+   na.omit() %>%
+   as.data.frame()
```

Antes del modelado, analizaremos la relación de los niveles del factor en la variable objetivo, lo que nos permitirá conocer el nivel NIR (No Information Rate, significa la mayor proporción de los niveles de clase en nuestra variable de respuesta) para especificar resultados de precisión significativos. Lo crearemos con la función treemap.

```
> #Treemap of factor
> library(treemap)
> df_tidy %>%
+   count(gasoline) %>%
+   mutate(label=paste(gasoline,scales::percent(n/sum(n)),sep = "\n")) %>%
+   treemap(
+     index="label",
+     vSize="n",
+     title="",
+     palette="Accent",
+     border.col=c("black"),
+     border.lwds=1,
+     fontcolor.labels="white",
+     fontface.labels=1,
+     inflate.labels=F
+ )
```



El diagrama anterior nos permite comprender, cualquier modelo que creemos debe tener una tasa de precisión superior al 65.7 % para ser significativo. El estadístico kappa es una medida de precisión eficaz en términos de la tasa de ausencia de información (65.7 %) que acabamos de mencionar.

$$\kappa = \frac{P_r(a) - P_r(e)}{1 - P_r(e)}$$

- $P_r(a)$ es la suma de las tasas de verdaderos positivos y verdaderos negativos (concordancia real).
- $P_r(e)$ es la concordancia esperada: (TP + FP)*(TP + FN) + (FP + TN)*(FN + TN).
- Utilizaremos **kappa no ponderado** como valor de medición. Dado que no existe correlación entre los niveles de los factores de la variable objetivo (**diferentes grados de concordancia**).

Podemos construir nuestros modelos y comparar los resultados de kappa para diferentes valores de semilla en un gráfico de plotly.

```
> library(adabag) #Boosting
> library(ipred) #Bagging
> library(caret) #Bagging control object
> library(vcd) #Kappa
> library(plotly) #interactive plot

> #Bagging
```

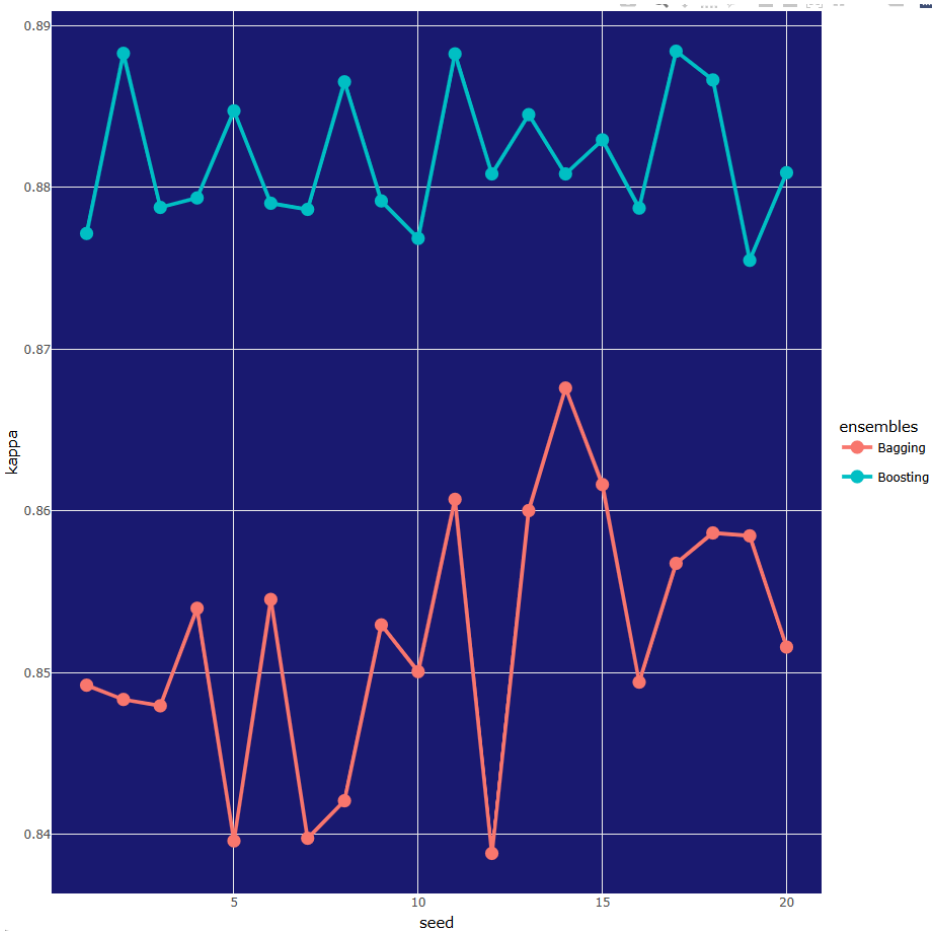
```
> ctrl <- trainControl(method = "cv", number = 10)
> kappa_bagg <-
+   lapply(
+   1:20,
+   function(x){
+   set.seed(x)
+   train(gasoline ~ .,
+     data = df_tidy,
+     method = "treebag",
+     trControl = ctrl)$results[["Kappa"]]}
+   ) %>%
+   unlist()

> #Boosting
> kappa_boost <-
+   lapply(
+   1:20,
+   function(x){
+     set.seed(x)
+     boosting.cv(gasoline ~ ., data = df_tidy) %>%
+     .$confusion %>%
+     Kappa() %>%
+     .$Unweighted %>%
+     .[[1]]}
+   ) %>%
+   unlist()

> #Kappa simulation on a plotly chart
> kappa_plot <-
+   data.frame(
+    seed=rep(1:20, 2),
+    kappa= c(kappa_bagg, kappa_boost),
+    ensembles=c(rep("Bagging", 20), rep("Boosting", 20))
+   ) %>%
+   ggplot(aes(x=seed, y=kappa, color=ensembles))+
+   geom_point(size=3)+
+   geom_line(size=1)+
+   theme_minimal()+
+   theme(panel.background = element_rect(fill = "midnightblue", color = NA),
+    panel.grid.minor.y = element_blank(),
+    panel.grid.minor.x = element_blank())

> ggplotly(kappa_plot) %>%
+  layout(legend=list(orientation="v", y=0.5))
```

Cuidado porque la imagen se despliega en un navegador web.



Como se puede ver arriba, el algoritmo de boosting parece ser levemente mejor que el bagging pero, en general, ambos algoritmos también tienen resultados de precisión muy alta.