Práctica Series de Tiempo, Agrupamiento, Clasificación

Ejercicio 1

Este ejercicio ha sido desarrollado en R. Para ello se ha decidido trabajar con el IDE RStudio y la base de datos PostgreSQL.

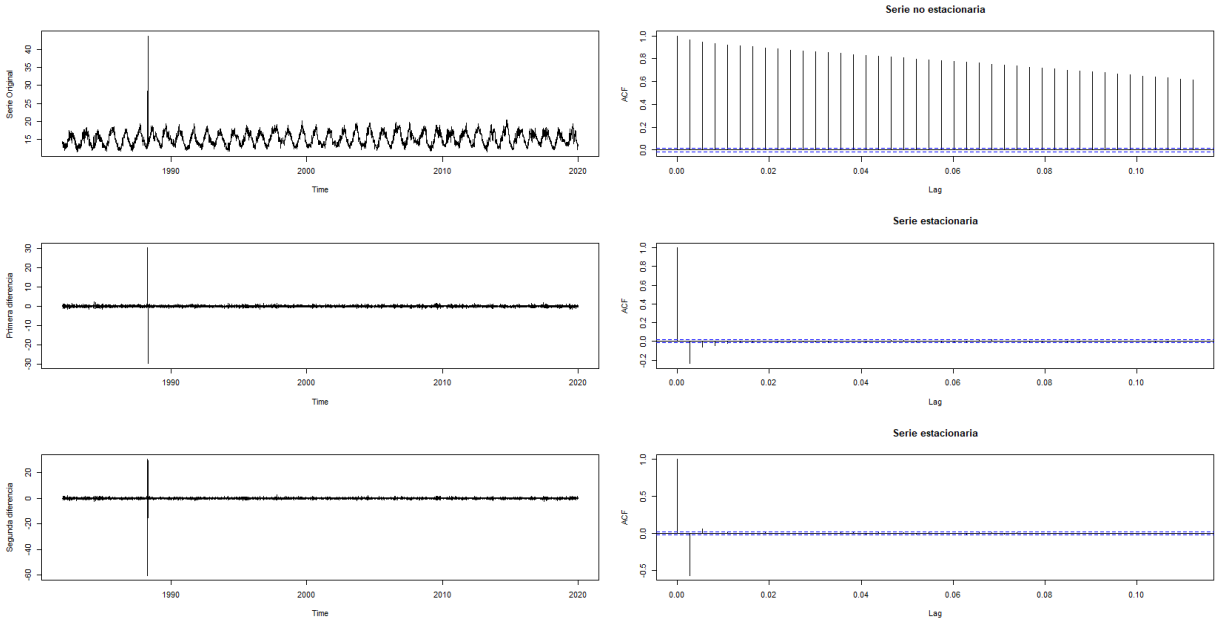
En primer lugar, se ha descargado el archivo .csv con las series temporales. En PostgreSQL mediante el gestor PgAdmin se ha creado la tabla para guardar los datos y a continuación insertar el csv en dicha tabla.

```
CREATE TABLE public.seriestemporales
(
marca_temporal timestamp,
ekmx double precision,
sst double precision
);
```

Una vez hecho esto, ya contamos con los datos sobre las series temporales guardados en nuestra base de datos, vamos a proceder a operar sobre ellos.

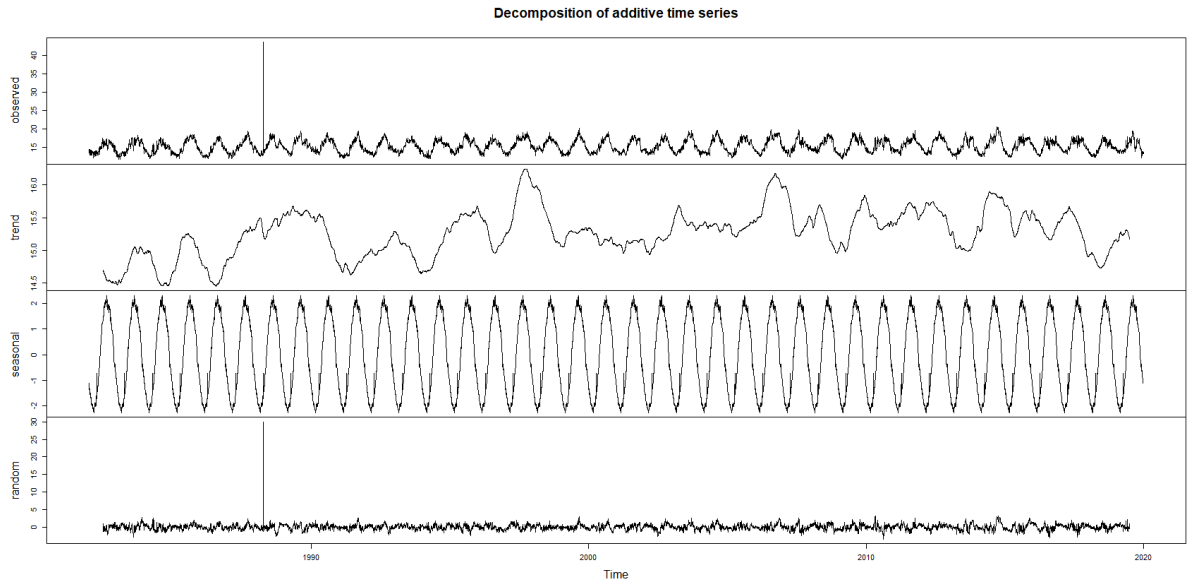
Como primer paso, nos deberemos conectar a Postgres desde RStudio, para eso instalamos el paquete correspondiente 'RPostgres', nos conectamos a la BD y realizamos la respectiva consulta sobre la tabla para recuperar los datos.

Ahora ya contamos con los datos en el entorno de RStudio. Comenzaremos con la serie de tiempos sobre la tercera columna (**sst**). Creamos la primera serie temporal con la función **ts()** de R y detectamos los datos atípicos de la serie temporal con la función **tsoutliers()** del paquete **forecast.** Para el cálculo de la primera y segunda diferencia realizaremos un diff sobre la serie temporal original y la primera diferencia respectivamente. Para el cálculo de la autocorrelación de cada una de las series temporales, utilizaremos la función **acf().** Los resultados obtenidos de las tres series temporales generadas y sus respectivas autocorrelaciones son las siguientes:

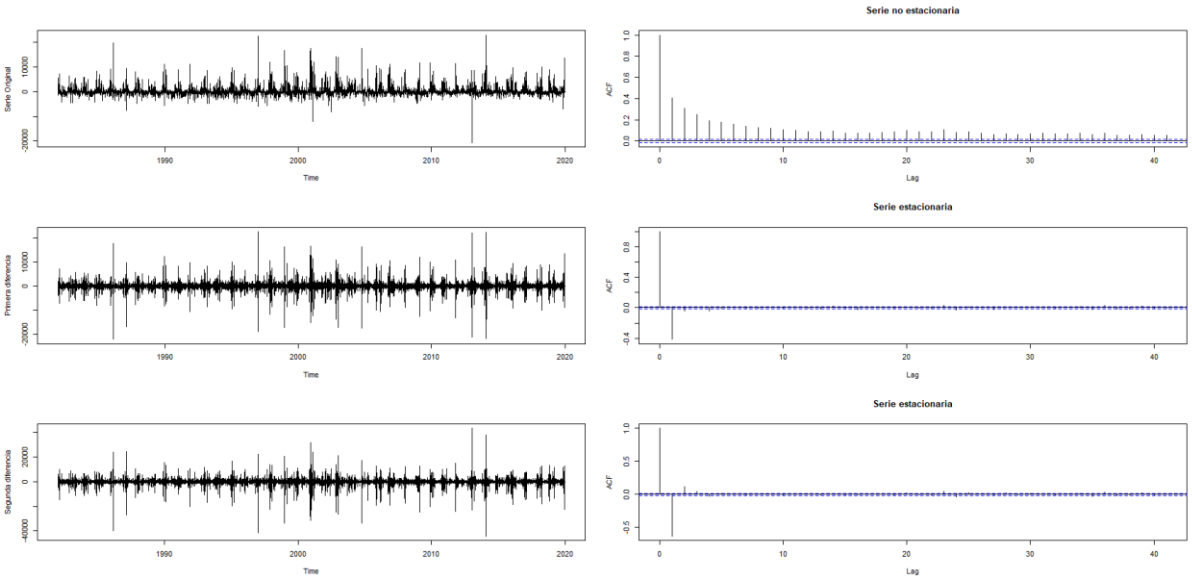


De los resultados anteriores, podemos observar como la gráfica ACF de la serie no estacionaria va decrementando su valor lentamente, mientras que las series no estacionarias mantienen sus gráficas de ACF próximas a cero todo el tiempo.

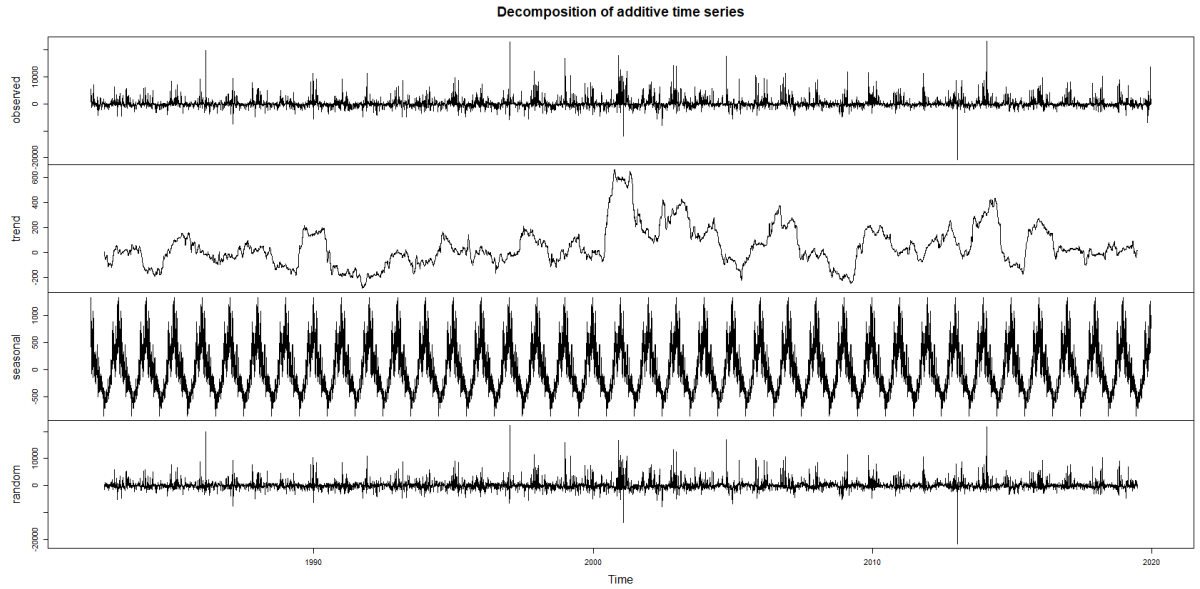
A continuación descomponemos la serie temporal en sus componentes estacionales y tendencia, para ello utilizamos el método decompose() sobre la variable donde guardamos la serie temporal. Graficamos los resultados:



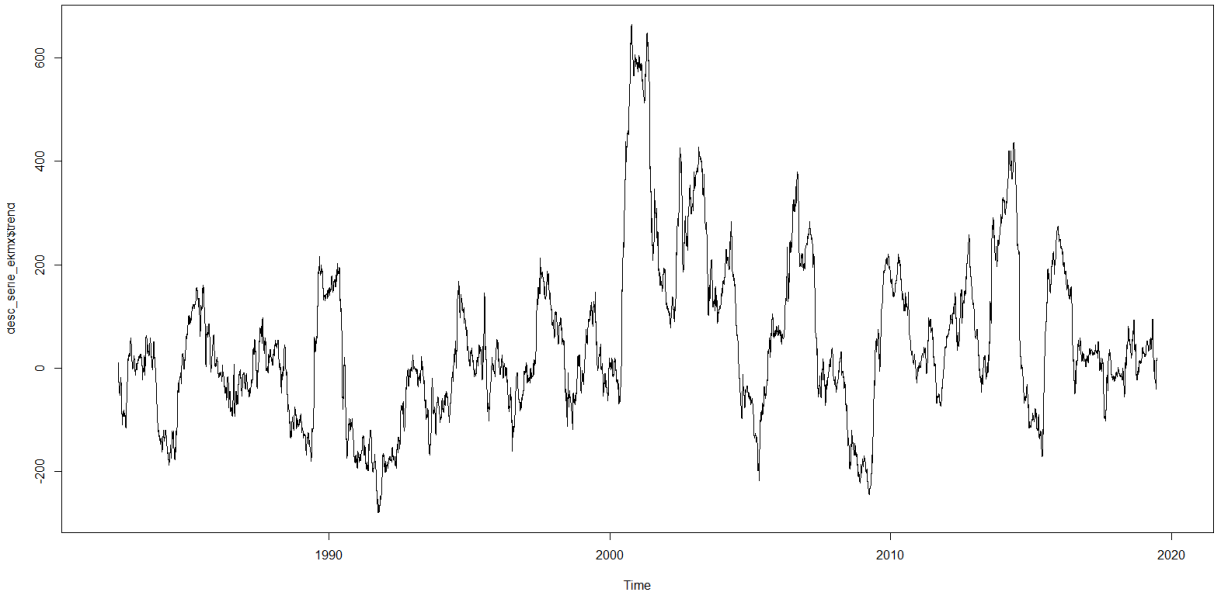
A continuación, procederemos de forma análoga a como lo hemos hecho para obtener los resultados para nuestra segunda variable ekmx. Resultado de la serie temporal original, las dos diferenciaciones y sus respectivas autocorrelaciones:



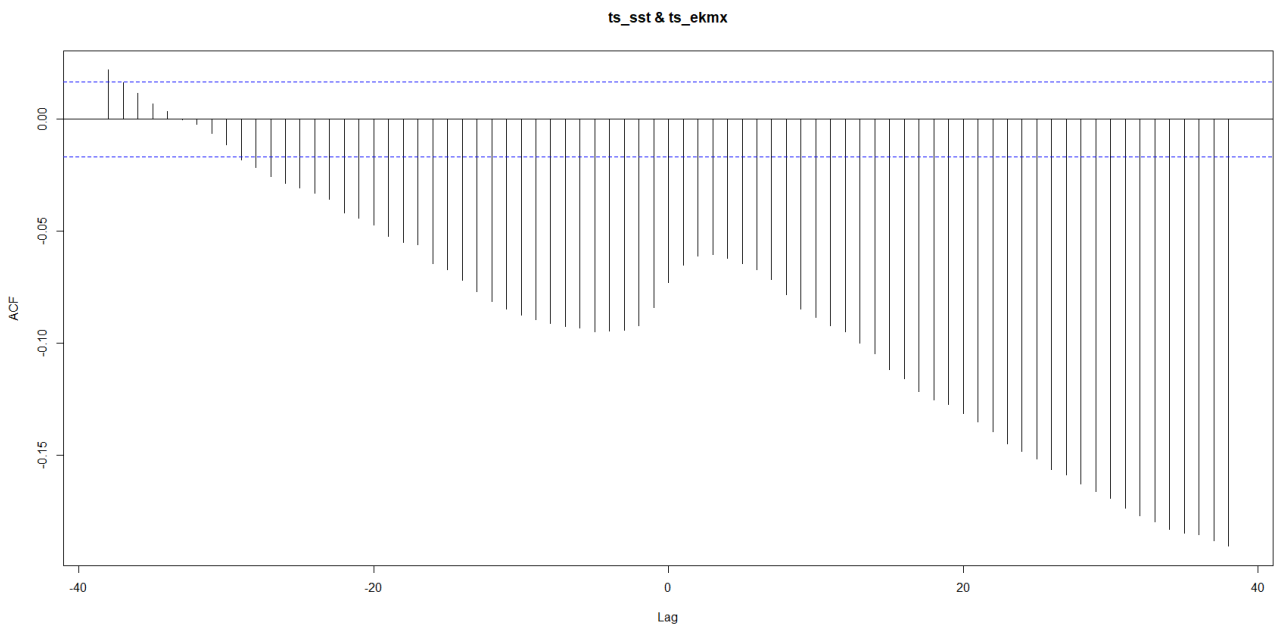
Resultados de la descomposición de esta segunda serie temporal, en este caso se han interpolado los valores NA con la función **na.approx()**:



Interpolamos linealmente el trend de la segunda serie temporal, utilizando na.approx() dentro de la función dexompose. El resultado obtenido es el siguiente:



Por último, se ha calculado la correlación cruzada entre las dos series temporales, para ello se ha utilizado la funcíon de R **ccf()**:



Ejercicio 2

Este ejercicio debía ser realizado en Python. Por ello, se ha desarrollado un Jupyter Notebook a través de Google Research Colab. Se ha creado una libreta compartida, y se ha escrito el código por celdas, facilitando su legibilidad. El PDF del Notebook está adjuntado en la práctica, pero se explica a continuación parte de la implementación.

Agrupamiento

Para la práctica de Agrupamiento, se trabaja con el 1% de los datos, ya que sino tardaba demasiado la ejecución. Se extrae una muestra indicando las columnas sobre las que se quiere trabajar, y se normalizan.

A continuación, se aplican los algoritmos correspondientes: K-Means, DBSCAN y SOM (con la librer-ia sklearn_som). Los resultados se visualizan con seaborn el cual da una representación muy clara de los grupos generados. Cabe mencionar que, en el caso de DBSCAN, por algún motivo no se generaban los grupos correspondientes, sino que devolvía -1 y 0 en los datos.

En cuanto a los resultados obtenidos, no se aprecia mucha diferencia entre K-Means y SOM, y dado que en tiempo de ejecución también es muy similar, es difícil escoger cual es la mejor de estas. Por lo tanto, es muy probable que dependa del tipo de problema.

Regresión

En este caso, se aplican los algoritmos de Random Forest Classifier y MLP Classifier (Multi-Layer Perception). El primer caso es, una vez más cargar los datos y seleccionar una muestra, esta vez del 0.5% (ya que estos algoritmos requieren de muchísimo tiempo de ejecución de entrenamiento).

El procedimiento es el mismo en ambos algoritmos: se separa el dataset en su X y en su Y, y se separan estos datos para generar uns conjuntos de entrenamiento y otros para testear. Se pasan los datos de entrenamiento a la función fit para entrenar los algoritmos, y a continuación con predict se intenta averiguar las Y de los datos de testeo.

Se adjunta, además, para el Random Forest Classifier, una imagen del árbol de decisión generado (se ha indicado que un máximo de 4 capas de profundidad, ya que sino el algoritmo nunca acababa de entrenar, con 4 capas necesita más de una hora). Ambos algoritmos necesitan de varias horas para

ejecutarse, y los resultados obtenidos no son muy buenos (precisión del 0.5%), pero esto se debe muy en parte a que no se pueden utilizar todos los datos del dataset debido a su elevado tiempo computacional y la falta de recursos para ejecutarlos.

Reglas de Asociación

Ejercicio

Este ejercicio ha sido realizado en Python mediante Jupyter Notebook. Para ello se ha creado una libreta en la que se ha dividido el código en distintas celdas, facilitando la legibilidad tanto del propio código como de las salidas del mismo. El PDF del Notebook está adjuntado en la práctica, pero se explica a continuación parte de la implementación, además de los resultado obtenidos y un poco de la base teórica en la que se sustenta dicha implementación.

En este ejercicio se intentará sacar para un conjunto de datos determinado sus reglas de asociación, las cuales sirven para encontrar relaciones entre los elementos de un conjunto de datos de la forma: X->Y, siendo X e Y subconjuntos de los ítems. En este caso el conjunto de datos se trata de una lista de compras realizadas, donde cada fila agrupa los ítems que se han comprado junto a los datos de identificación de la propia compra. Entonces, las reglas de asociación permitirán saber qué ítems suelen comprarse a la vez y cúales condicionan a la compra de otros.

Sin embargo, a esto hay que añadirle un problema, y es que se pueden generar un gran número de reglas de asociación, a pesar de que no todas sean interesantes debido al bajo porcentaje de aparaciones que tendrán algunas a lo largo del conjunto de datos, por lo que se deberá añadir algún tipo de filtro a la hora de que se generen las asociaciones. Para realizar este filtrado se deberán tener en cuenta las siguientes medidas:

- *Soporte*: indica la frecuencia con la que un subconjunto de ítems aparece en las distintas transacciones del conjunto de datos. Para una regla de asociación, sería el número de entradas que contienen X e Y dividido por el número total.
- *Confianza*: indica la frecuencia con la que se ha encontrado que la regla de asociación es cierta. Para una regla de asociación, sería la medida de la frecuencia en la que Y aparece en las transacciones que contienen X.
- *Lift*: es una medida que combina soporte y confianza para indicar el grado de in/dependencia en la aparición de un subconjunto de items respecto de otro. Para una regla de asociación, una de las formas de calcularlo es la confianza de X e Y entre el soporte de Y.

Ahora ya pasamos al código en sí. En primer lugar, deberemos cargar el conjunto de datos y transformarlos para cuando toque generar las asociaciones. La carga de datos, *celda 1*, la realizaremos con la librería de python *pandas* de forma que el contenido de cada fila quede en un único campo y así se puedan separar después de forma manual. A continuación, se procede a separar el dataset en diferentes campos, para ello separamos por el delimitador ‘,’ en sus dos primeras apariciones, de forma que los ítems de la compra se agrupen en un único campo. En la salida de los comandos ejecutados en la *celda 1* se puede observar que hay un total de 14963 filas de datos y 3 campos: *Usuario*, *Fecha* y *Item*, pero el realmente interesante es el último, que agrupa los ítems de cada compra.

Sin embargo, hay que seguir haciendo transformaciones sobre la lectura de los datos antes de generar las asociaciones, ya que lo que nos interesa es que estas se generen teniendo en cuenta solo los ítems de las compras, por lo que crearemos una lista que guarde cada lista de ítems de cada compra (*celda 2*).

Ahora sí, en la *celda 3*, ya se crean las reglas de asociación. Para ello se utiliza el objeto *apriori* del paquete de python *apyori*. Al crear un objeto de este estilo, lo que estamos haciendo es ejecutar el algoritmo apriori para sacar las reglas de asociación sobre un conjunto de datos determinado. En la creación del objeto *apriori* se pasan los siguientes parámetros (pero no son los únicos que admite):

- El *conjunto de datos* de donde se van a extraer las reglas de asociación. En este caso, la lista de listas con los ítems de compra que se creó anteriormente.
- *min_support*: es el soporte mínimo que deben tener las asociaciones que se van a extraer. En este caso su valor es 0.001, un valor bajo porque la probabilidad de que un subconjunto de ítems se hayan comprado a la vez es muy baja dada la diversidad y el tamaño del conjunto de datos, pero no mucho para que la compra de estos elementos no sean de un caso aislado.
- *min_confidence*: es la confianza mínima que deben tener las asociaciones que se van a extraer. En este caso su valor es 0.1, un valor bajo por la misma razón que en el parámetro anterior, pero no mucho para encontrar una fuerte relación entre la compra de un subconjunto de ítems y otro.
- *min_lift*: es el lift mínimo que deben las asociaciones que se van a extraer. En este caso su valor es 1.25, un valor bajo por la misma razón que en los parámetros anteriores, pero superior a 1 para indicar que este conjunto aparece una cantidad de veces superior a lo esperado bajo condiciones de independencia, así como evitar las apariciones de listas vacías en la precedencia (por lo que observe al jugar con los parámetros).

En relación a los valores escogidos para cada parámetro, cabe destacar que se fueron haciendo pruebas variando los parámetros e intentando sacar una lista pequeña de asociaciones y con valores para los parámetros que tuvieran sentido dentro de la naturaleza del problema.

Por último, en la *celda 4*, se añade formato a los datos anteriormente generados y se imprimen por pantalla. En la salida se observa que se han generado un total de 11 asociaciones para los parámetros anteriormente especificados. Se puede observar que la asociación con mayor soporte es la de *queso de gratinar -> bollos* apareciendo un 0.00147, la de mayor confianza la de *salchichas y bollos -> leche entera* con un 0.2125 y la de mayor lift la de *salchichas y leceh entera -> yogurt* con un 1.91176. Ahora con este pequeño estudio se puede analizar qué ítems de la compra deberían estar próximos para aumentar las ventas.

Anexos

Se adjuntan 2 PDFs para los siguientes apartados:

- 1_SeriesTemporales.r: Ejercicio 1
- 2_RegressionClustering.pdf: Ejercicio 2
- 3_ Asociacion.pdf: Reglas de Asociación (Ejercicio 3)