DETECCIÓN DE ANOMALÍAS CON CADENAS DE MARKOV

Desarrollo de Software Crítico

CRISTINA BARÓN SUÁREZ
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Universidad de Málaga

Índice

Imágenes

Entrenamiento y detección de anomalías en ficheros modificados

En la clase `Main` se crea el método `leerFichero`, que permitirá leer de un fichero los dispositivos con sus correspondientes timestamps y medidas. Mientras se leen, se irán almacenando en un HashMap los dispositivos (keys) y su lista de medidas asociada (values). Para ello, se crea una nueva clase: `DeviceDict`

- **Método de la clase principal para leer de fichero**

```
public static void leerFichero() throws IOException {
    // Abre un fichero en modo lectura
    FileReader file = new FileReader("training.txt ");
    BufferedReader br = new BufferedReader(file);

    // Obtiene cada linea hasta el fin del fichero
    String line;
    while((line = br.readLine()) != null) {
        // Divide las lineas en palabras (delimitador = ",")
        String words[] = line.split(",");
        boolean existe = (DeviceDict.get(words[0]) != null);
        if(existe) {
            List<Integer> listDevice = DeviceDict.get(words[0]);
            listDevice.add(Integer.parseInt(words[2]));
            DeviceDict.put(words[0], listDevice);
        }else { // El dispositivo aún no está reconocido
            List<Integer> li = new ArrayList<>();
            li.add(Integer.parseInt(words[2])); // measurement
            DeviceDict.put(words[0], li);
        }
    }
    br.close();
}
```

Ilustración 1. Leer fichero de entrenamiento.

- **Clase DeviceDict**

```
public class DeviceDict {
                        //  Device   Medidas
    protected static HashMap<String, List<Integer>> hm = new HashMap<>();

    public static void put(String variable, List<Integer> valor) {
        hm.put(variable, valor);
    }

    public static List<Integer> get(String variable) {
        List<Integer> valor = hm.get(variable);
        return valor;
    }

}
```

Ilustración 2. Clase DeviceDict

Tras conseguir almacenar los datos y con el fin de crear la matriz de probabilidades, se crea una nueva clase llamada `Matriz`. En esta se declararán los métodos que permitirán hallar dicho array bidimensional. Los métodos son los que siguen a continuación.

```
private static int numIntervalo = 10;
private static int rangoVentana = 4;
```

- **Método para obtener las medidas.**

  Obtiene las medidas de entre todas las máquinas, insertando un "-1" cada vez que se cambie de dispositivo. Esto se realiza debido a que no se debe estudiar la transición entre la última medida de un dispositivo y la primera del siguiente.

```
public static List<Integer> obtenerValores(){
    List<Integer> valores = new ArrayList<>();
    for(String device : DeviceDict.hm.keySet()) {
        List<Integer> li = DeviceDict.get(device);
        for(int i = 0; i < li.size(); i++) {
            valores.add(li.get(i));
        }
        valores.add(-1);     // Para saber cuándo se cambia de máquina
    }
    return valores;
}
```

Ilustración 3. Obtener medidas para el entrenamiento

Por otro lado, he discretizado los valores decidiendo que la matriz tenga un tamaño de 10x10. Es decir, las mediciones se agruparán en intervalos iguales de tal modo que entre todas queden distribuidas en 10 distintas filas y columnas. Para ello, se necesitará obtener el valor más grande y pequeño de entre todas las mediciones, lista que se recibe por parámetro en los dos métodos. Como es evidente, en ambos casos se ignora el cambio de máquina ya que esta, recordemos, no es una medición real.

- **Método para obtener el mayor de entre todas las medidas.**

```
public static int mayor(List<Integer> valores) {
    int mayor = Integer.MIN_VALUE;
    for(int i = 0; i < valores.size(); i++) {
        int n = valores.get(i);
        if(n != -1 && mayor < n) {  // Cambio de maquina
            mayor = n;
        }
    }
    return mayor;
}
```

Ilustración 4. Obtener la mayor medida del entrenamiento

- **Método para obtener el menor de entre todas las medidas.**

```
public static int menor(List<Integer> valores) {
    int menor = Integer.MAX_VALUE;
    for(int i = 0; i < valores.size(); i++) {
        int n = valores.get(i);
        if(n != -1 && n < menor) {  // Cambio de maquina
            menor = n;
        }
    }
    return menor;
}
```

Ilustración 5. Obtener la menor medida del entrenamiento

- **Método para calcular el estado de una medida**

En este método, se le pasa por parámetro los valores mayor y menor calculados previamente durante el entrenamiento y una medida. Y, como se ha descrito anteriormente, se estudia a cuál de los 10 intervalos debe pertenecer. Este estado es el que se devuelve.

Además, también se contempla que puedan llegar valores mayores y menores que los que fueron estudiados. Estas mediciones se incluirán en la primera y última posición del array bidimensional, según corresponda.

```
// Discretizamos en valores de 10:
static int calcEstado(int menor, int mayor, int n) {
    float div = (float)(n-menor)/(float)(mayor-menor);
    int aux = Math.round((numIntervalo-1)*div);
    if(aux < 0) {    // Por si llegan valores menores que el menor
        aux = 0;
    }else if(aux > (numIntervalo-1)) {  // Por si llegan valores mayores que el mayor
        aux = numIntervalo-1;
    }
    return aux;
}
```

Ilustración 6. Calcular el estado de una medida

- **Método para construir la matriz de transiciones**

Para saber qué probabilidad tiene una transición de ocurrir es primordial saber la cantidad de ocurrencias que tiene dicha transición. Para ello, con el método anterior se estudia el estado que dos mediciones consecutivas tienen y en la casilla resultante se le incrementa en uno el número de ocurrencias. Como venimos haciendo anteriormente, el cambio de máquina se obvia y evitamos que la última y la primera medición de dos máquinas consecutivas se consideren como una transición.

Recibe por parámetro el menor y mayor valor y una lista de mediciones.

```
public static double[][] construirMatriz(int menor, int mayor, List<Integer> valores) {
    // Creamos la estructura de datos
    double[][] matriz = new double[numIntervalo][numIntervalo];
    for(int i = 0; i < valores.size()-1; i++) {
        if(valores.get(i) != -1 && valores.get(i+1) != -1) {    // Cambio de maquina
            int val1 = calcEstado(menor, mayor, valores.get(i));    // Fila = Desde donde viene
            int val2 = calcEstado(menor, mayor, valores.get(i+1));  // Columna = A donde va
            matriz[val1][val2]++;
        }
    }
    return matriz;
}
```

Ilustración 7. Construcción de la matriz N

- **Método para construir la matriz de probabilidades**

Al igual que en el método anterior, se recibe por parámetro el menor y mayor valor y la lista de mediciones. Con estos datos, se calcula la matriz de transiciones y se crea otro array bidimensional del mismo tamaño que esta. Esta matriz almacenará en cada casilla la probabilidad de que cierta transición ocurra. Para ello, se divide la cantidad de ocurrencias que ha habido de dicha transición entre la suma total de todas las transiciones que se dieron en la que dicho primer valor era el antecesor a otra medida consecutiva. Siempre considerando que cuando hablamos de “valor” hablamos de un conjunto de mediciones agrupadas en un cierto intervalo, como se ha explicado previamente.

En el caso de que una transición no esté contemplada, en lugar de considerar una probabilidad imposible, se le asignará una probabilidad del 0’5.

```
public static double[][] construirMatrizProb(int menor, int mayor, List<Integer> valores){
    double[][] matriz = construirMatriz(menor, mayor, valores);
    // Se suman todos los valores de la fila para probar la probabilidad
    double[][] matrizProb = new double[numIntervalo][numIntervalo];
    for(int fil = 0; fil < matriz.length; fil++) {
        double sumFila = 0;
        for(int col = 0; col < matriz.length; col++) {
            if(matriz[fil][col] == 0) {
                // Si algo no ha ocurrido, se le asigna una frecuencia baja:
                matriz[fil][col] = 0.5;
            }
            sumFila += matriz[fil][col];
        }
        for(int col = 0; col < matriz.length; col++) {
            matrizProb[fil][col] = matriz[fil][col]/sumFila;
        }
    }
    return matrizProb;
}
```

Ilustración 8. Construcción de la matriz de probabilidades

- **Método para calcular el punto de corte**

  Una vez se dispone de la matriz de probabilidades, se desea estudiar qué ventana de 3 transiciones (es decir, 4 mediciones) ha sido la que ha ofrecido la probabilidad más baja de ocurrencia. Dicho porcentaje será el que la función devuelva, considerándose el punto de corte. Es decir, cualquier ventana de 3 con una probabilidad igual o inferior a dicho porcentaje se considerará anómala. Esto será útil cuando se introduzcan nuevas mediciones.

  Para calcular la probabilidad de una ventana de 3 transiciones se calcula el estado de cada medida y, por cada dos consecutivos, se estudia qué porcentaje devuelve la matriz de probabilidades para dichos valores. Estas probabilidades se insertan en una lista que cuando tenga un tamaño de 4 multiplicará todo su contenido y el resultado será la probabilidad de que dicha ventana se diese. El primer valor de la lista se eliminará una vez se conozca la probabilidad de la ventana y se haya estudiado si esta es menor que la considerada punto de corte hasta el momento para dar paso al estado del siguiente valor que corresponda estudiar en una nueva ventana de 4 estados.

  Al igual que en el método anterior, se recibe por parámetro el menor y mayor valor y la lista de mediciones.

```
/* 1. Calcular estado del valor
   2. Cuando se tengan 4 estados
   3. Multiplicar las probabilidades de las 3 transiciones anteriores
   4. Obtener la menor probabilidad */
public static double puntoCorte(int menor, int mayor, List<Integer> valores) {
    double[][] matrizProb = construirMatrizProb(menor, mayor, valores);
    List<Integer> estados = new ArrayList<>();
    double menorP3T = Double.MAX_VALUE;
    for(int i = 0; i < valores.size(); i++) {
        if(valores.get(i) != -1) {  // Para no considerar el cambio de máquina
            estados.add(calcEstado(menor, mayor, valores.get(i)));
            if(estados.size() == rangoVentana) {
                double p3t = 1.0;
                for(int j = 1; j < estados.size(); j++) {
                    p3t *= matrizProb[estados.get(j-1)][estados.get(j)];
                }
                if(p3t < menorP3T) {
                    menorP3T = p3t; // Obtener menor probabilidad
                }
                estados.remove(0);
            }
        }

    }
    return menorP3T;
}
```

Ilustración 9. Calcular el punto de corte

Persistencia y detección de anomalías “online”

Con el fin de poder detectar las anomalías en tiempo real, se almacenará en un fichero la matriz de probabilidades y el punto de corte. Las funciones que siguen a continuación se encuentran en la clase `Matriz`.

- **Método para insertar la matriz de probabilidades en un fichero**

  Se recibe por parámetro el menor y mayor valor y la lista de mediciones de entrenamiento.

```
public static void matrixP2file(int menor, int mayor, List<Integer> valores) {
    double[][] matriz = construirMatrizProb(menor, mayor, valores);
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter("matrizP.txt"));
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz.length; j++) {
                    bw.write(matriz[i][j] + "\t"); // delimitador de columnas = "\t"
            }
            bw.newLine();
        }
        bw.flush();
        bw.close();
    } catch (IOException e) {}
}
```

Ilustración 10. Escribir la matriz de probabilidades en un fichero

- **Método para insertar el punto de corte en un fichero**

  Se recibe por parámetro el menor y mayor valor y la lista de mediciones de entrenamiento.

```
public static void PC2file(int menor, int mayor, List<Integer> valores) {
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter("puntoCorte.txt"));
        double n = puntoCorte(menor, mayor, valores);
        bw.write(String.valueOf(n));     // Lo guarda como String
        bw.flush();
        bw.close();
    } catch (IOException e) {}
}
```

Ilustración 11. Insertar el punto de corte en un fichero

Posteriormente, estos ficheros se cargan en el programa para poder detectar anomalías "online". Para ello se ha creado una nueva clase `DetectorAnomalias`, que está compuesta por las funciones que se mostrarán a continuación.

```
// Mismos valores que en la clase Matriz:
private static int rangoVentana = 4;
private static int numIntervalo = 10;
```

- **Método que lee la matriz de probabilidades de fichero**

```
public static double[][] leerMatrizP() throws java.io.FileNotFoundException{
    Scanner sc = new Scanner(new File("MatrizP.txt"));
    double[][] mp = new double[numIntervalo][numIntervalo];

    int fil = 0;
    while(sc.hasNextLine()) {
        String[] numsFila = sc.nextLine().split("\t");
        int col = 0;
        for(int i = 0; i < numIntervalo; i++) {
            mp[fil][col] = Double.parseDouble(numsFila[i]);
            col++;
        }
        fil++;
    }
    sc.close();
    return mp;
}
```

Ilustración 12. Leer la matriz de probabilidades de un fichero

- **Método que lee el punto de corte de fichero**

```
public static double leerPuntoCorte() throws java.io.FileNotFoundException{
    Scanner sc = new Scanner(new File("puntoCorte.txt"));
    double pc = Double.parseDouble(sc.nextLine());
    sc.close();
    return pc;
}
```

Ilustración 13. Leer el punto de corte de un fichero

- **Método para leer de teclado**

  Las mediciones a estudiar se leerán de teclado y hasta que no se inserte la palabra "FIN" no se dejará de leer nuevos valores. Estos se insertan en una lista de enteros, que es la que devuelve la función.

```
public static List<Integer> leerTeclado(){
    Scanner in = new Scanner(System.in);
    System.out.println("Se deben leer al menos 4 números de teclado");
    System.out.println("\"FIN\" para terminar de escribir en la consola");
    List<Integer> numeros = new ArrayList<>();
    boolean salir = false;
    while(!salir) {
        String s = in.nextLine();
        if(s.toUpperCase().equals("FIN")) {
            if(numeros.size() < 4) {
                System.out.println("Se deben leer al menos 4 números de teclado");
            }else {
                salir = true;
            }
        }else {
            int n = (int) Math.round(Double.parseDouble(s));
            numeros.add(n);
        }
    }
    in.close();
    return numeros;
}
```

Ilustración 14. Leer medidas desde teclado

- **Método para detectar anomalías**

  Esta función recibe por parámetro el menor y mayor valor detectados durante el entrenamiento. Además, actúa de manera similar al método que averigua el punto de corte en la clase `Matriz`:

  Se calcula la probabilidad de una ventana de 3 transiciones de la forma en que se explicó en el método mencionado anteriormente y si dicha probabilidad resultante es menor que el punto de corte establecido se considera que se ha detectado una anomalía.

```
public static void detectarAnomalia(int menor, int mayor) throws FileNotFoundException {
    List<Integer> nums = leerTeclado();
    detector(menor, mayor, nums);
}

public static void detector(int menor, int mayor, List<Integer> nums) throws FileNotFoundException {
    double pc = leerPuntoCorte();
    double[][] matrizProb = leerMatrizP();
    List<Integer> estados = new ArrayList<>();
    boolean anomalia = false;
    System.out.println("Punto de corte: " + pc);
    for(int i = 0; i < nums.size(); i++) {
        estados.add(Matriz.calcEstado(menor, mayor, nums.get(i)));
        if(estados.size() == rangoVentana) {
            double p3t = 1.0;
            for(int j = 1; j < estados.size(); j++) {
                p3t *= matrizProb[estados.get(j-1)][estados.get(j)];
            }
            if(p3t <= pc) {
                anomalia = true;
                System.out.println("Anomalía en la ventana de mediciones: " + nums.get(i-3)
                + " - " + nums.get(i-2) + " - " + nums.get(i-1) + " - " + nums.get(i) +
                "\n\tProbabilidad: " + p3t);
            }
            estados.remove(0);
        }
    }
    if(!anomalia) {
        System.out.println("No se han detectado anomalias");
    }
}
```

Ilustración 15. Detectar anomalías

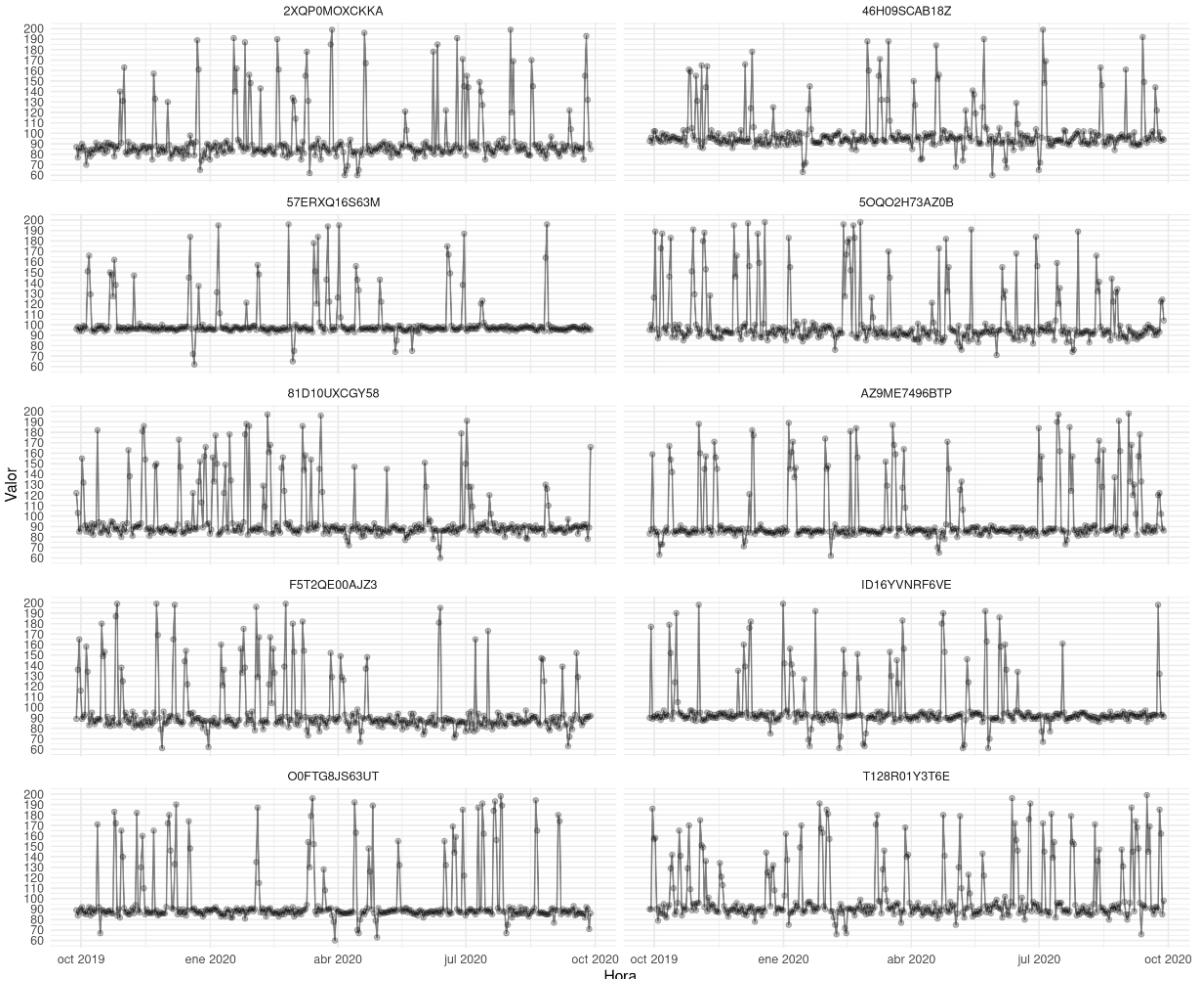
Justificante de actuación

He empleado R, lenguaje de programación visto en la asignatura de 1º curso Métodos Estadísticos para la Computación, con el fin de observar la forma que tienen los datos y justificar el razonamiento llevado a cabo en Java. Los datos en Java se han ido comparando para que den el mismo resultado que los que se muestran a continuación.

Lectura de datos

Tras la lectura del fichero de entrenamiento, para cada dispositivo se ha creado una gráfica con sus correspondientes mediciones con el fin de poder observar la forma que tienen los datos.

```
ggplot(df,aes(x=Hora,y=Valor))+
  geom_point(alpha=0.25)+
  geom_line(alpha=0.5)+theme_minimal()+
  scale_y_continuous(breaks=seq(60,200,by=10))+
  facet_wrap(~Device,ncol = 2)
```



Categorización de las transiciones

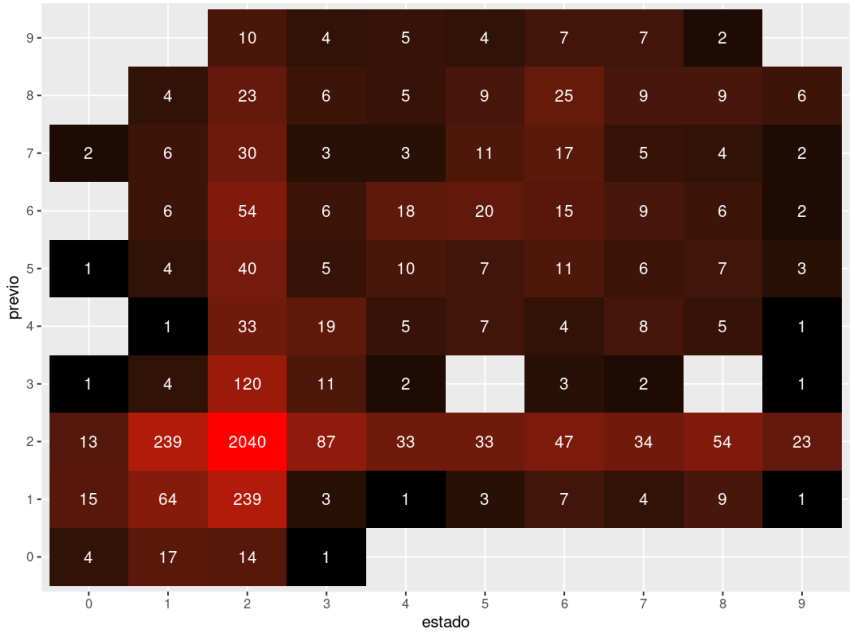
Para estudiar la frecuencia de las transiciones más habituales se dividen las observaciones en una serie de grupos, entre la observación menor y la mayor, para asegurarnos que la probabilidad de cada transición tiene suficientes observaciones. El número de intervalos se asigna a 10.

Para cada valor observado se calcula su estado. Además, también se mostrará su estado previo, que se colocará en una columna a su lado. Se muestran solo 15 líneas:

Device	Hora	Valor	estado	previo
T128R01Y3T6E	1569596235.00	90	2	NA
T128R01Y3T6E	1569682197.00	90	2	2
T128R01Y3T6E	1569769358.00	186	8	2
T128R01Y3T6E	1569855875.00	157	6	8
T128R01Y3T6E	1569942649.00	158	6	6
T128R01Y3T6E	1570028437.00	90	2	6
T128R01Y3T6E	1570113828.00	79	1	2
T128R01Y3T6E	1570201282.00	92	2	1
T128R01Y3T6E	1570287142.00	86	2	2
T128R01Y3T6E	1570374914.00	86	2	2
T128R01Y3T6E	1570461089.00	95	2	2
T128R01Y3T6E	1570546754.00	84	2	2
T128R01Y3T6E	1570633714.00	82	1	2
T128R01Y3T6E	1570719733.00	92	2	1
T128R01Y3T6E	1570805147.00	94	2	2

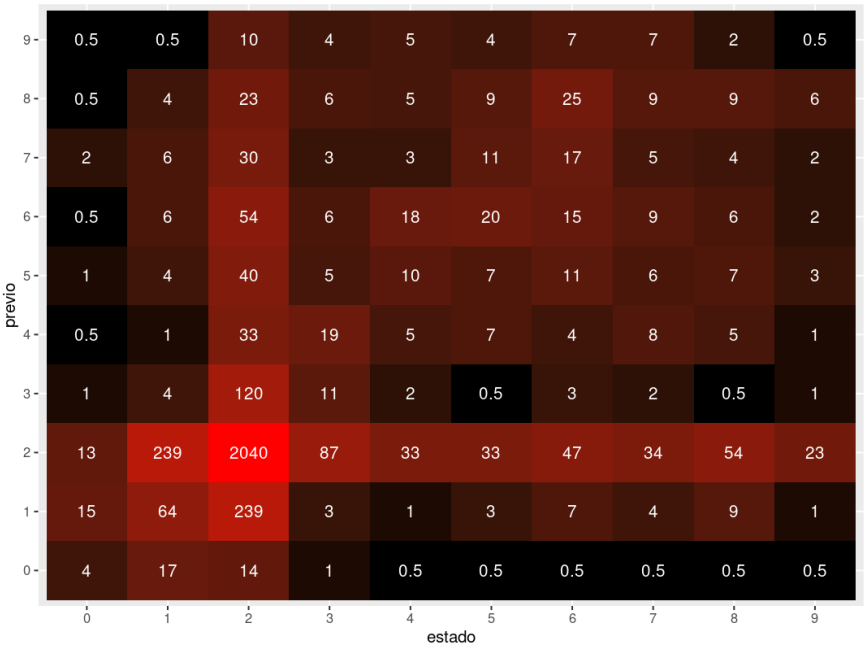
A continuación, se va a mostrar en un mapa de color las frecuencias de cada transición.

```
renderPlot({
ggplot(dfN() %>% mutate(log_n=log(n)),
       aes(y=previo, x=estado)) + geom_tile()+
  geom_tile(aes(fill = log_n)) +
    geom_text(aes(label = n),color="white") +
    scale_fill_gradient(low = "black", high = "red")+
  theme(legend.position="none")
},width = 800,height = 600,res=100)
```



Se puede observar que hay muchas transiciones que no se han presentado. Para no tratarlas como de probabilidad cero se les va a asignar una frecuencia de aparición baja: 0'5.

```
renderPlot({
ggplot(dfNCompletado() %>% mutate(log_n=log(n)),
       aes(y=previo, x=estado)) + geom_tile()+
  geom_tile(aes(fill = log_n)) +
    geom_text(aes(label = n),color="white") +
    scale_fill_gradient(low = "black", high = "red") +
  theme(legend.position="none")
},width = 800,height = 600,res=100)
```



A continuación, se va a calcular las probabilidades de cada transición condicionadas al estado previo. Para ello, se divide cada frecuencia, `n`, por el total de la fila, `N`. Se muestran únicamente las primeras filas:

previo	estado	n	N	p1t
0	0	4.00	39.00	0.10
1	0	15.00	346.00	0.04
2	0	13.00	2603.00	0.00
3	0	1.00	145.00	0.01
4	0	0.50	83.50	0.01
5	0	1.00	94.00	0.01
6	0	0.50	136.50	0.00
7	0	2.00	83.00	0.02
8	0	0.50	96.50	0.01
9	0	0.50	40.50	0.01
0	1	17.00	39.00	0.44
1	1	64.00	346.00	0.18
2	1	239.00	2603.00	0.09
3	1	4.00	145.00	0.03
4	1	1.00	83.50	0.01

En formato de matriz se vería como:

```
renderPlot({
ggplot(dfP() %>% mutate(log_p=log(p1t)),
       aes(y=previo, x=estado)) + geom_tile()+
#  scale_fill_viridis_c()+
  geom_tile(aes(fill = log_p)) +
    geom_text(aes(label = round(p1t,4)),color="white") +
    scale_fill_gradient(low = "black", high = "red")+
  theme(legend.position="none")
},width = 800,height = 600,res=80)
```

Cálculo de probabilidades de series consecutivas de transiciones

A continuación, se va a escribir, para secuencias de diferente longitud, la probabilidad de llegar al estado actual a partir de los estados anteriores. Se mostrarán únicamente las 10 mediciones.

- p1t → Probabilidad de la transición actual (implica dos estados consecutivos).
- p2t → Probabilidad de la transición actual y la anterior (implica tres estados consecutivos).
- p3t → Probabilidad de la transición actual y las dos anteriores (implica cuatro estados consecutivos).
- p4t → Probabilidad de la transición actual y las tres anteriores (implica cinco estados consecutivos).
- p5t → Probabilidad de la transición actual y las cuatro anteriores (implica seis estados consecutivos).
- p6t → Probabilidad de la transición actual y las cinco anteriores (implica siete estados consecutivos).

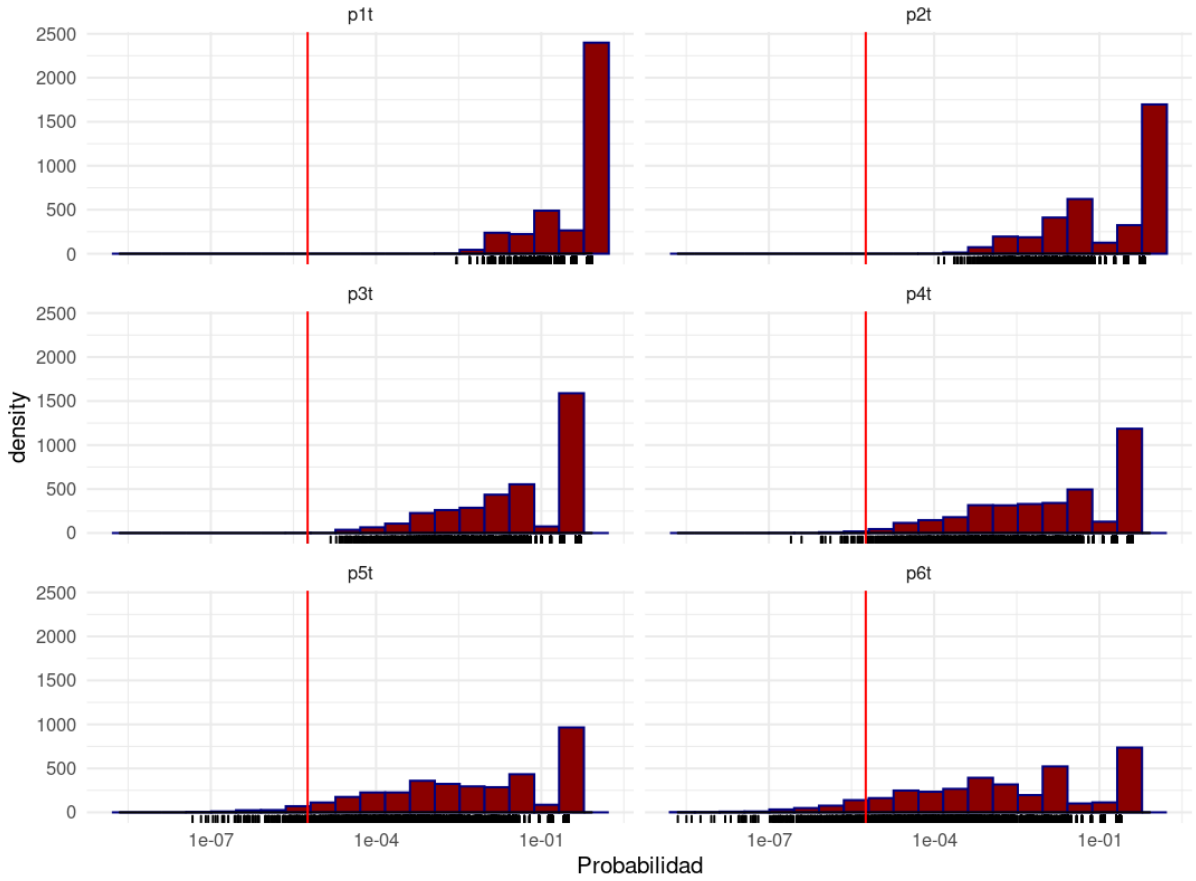
```
renderTable({
dfFinal()%>% head(10)
})
```

Device	Hora	Valor	estado	previo	n	p1t	p2t	p3t	p4t	p5t	p6t	pElegida
T128R01Y3T6E	1569596235.00	90	2	NA	NA	NA	NA	NA	NA	NA	NA	NA
T128R01Y3T6E	1569682197.00	90	2	2	2040.00	0.78	NA	NA	NA	NA	NA	NA
T128R01Y3T6E	1569769358.00	186	8	2	54.00	0.02	0.02	NA	NA	NA	NA	NA
T128R01Y3T6E	1569855875.00	157	6	8	25.00	0.26	0.01	0.00	NA	NA	NA	0.00
T128R01Y3T6E	1569942649.00	158	6	6	15.00	0.11	0.03	0.00	0.00	NA	NA	0.00
T128R01Y3T6E	1570028437.00	90	2	6	54.00	0.40	0.04	0.01	0.00	0.00	NA	0.01
T128R01Y3T6E	1570113828.00	79	1	2	239.00	0.09	0.04	0.00	0.00	0.00	0.00	0.00
T128R01Y3T6E	1570201282.00	92	2	1	239.00	0.69	0.06	0.03	0.00	0.00	0.00	0.03
T128R01Y3T6E	1570287142.00	86	2	2	2040.00	0.78	0.54	0.05	0.02	0.00	0.00	0.05
T128R01Y3T6E	1570374914.00	86	2	2	2040.00	0.78	0.61	0.42	0.04	0.02	0.00	0.42

Se observa la distribución de las probabilidades para un número de transiciones entre 1 y 6:

```
renderPlot({
dfFinal() %>% select(Device,ends_with("t")) %>% pivot_longer(cols = -c(Device)) %>%
  filter(complete.cases(.)) %>% select(-Device) %>%
  rename(Probabilidad=value) %>%
  ggplot(aes(x=Probabilidad))+geom_histogram(bins=20,color="navyblue",fill="darkred")+
    geom_density()+
     geom_rug()+
    geom_vline(xintercept = pc(),color="red")+
  facet_wrap(~name,ncol = 2)+
  scale_x_log10()+theme_minimal()
},width = 800,height = 600,res=100)
```

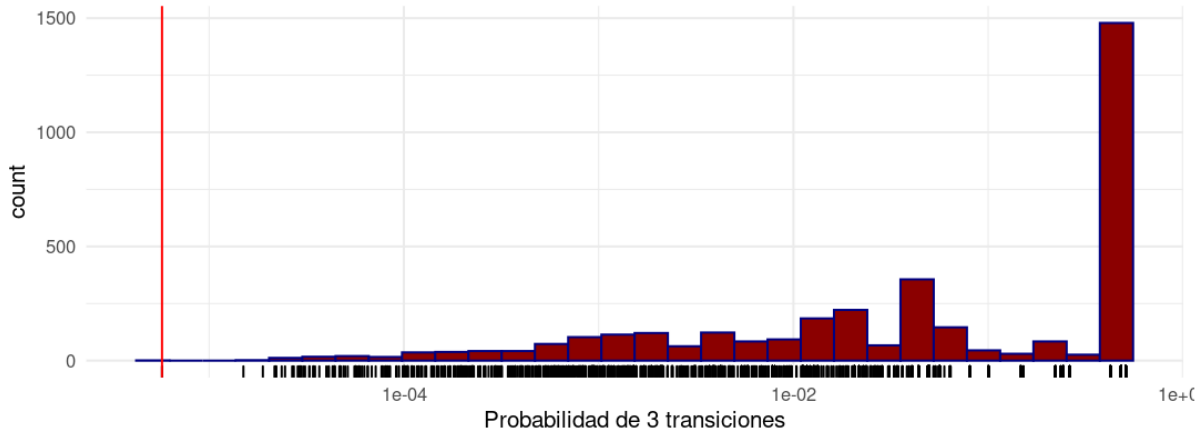
En las gráficas se muestra en una línea roja dónde se encuentra el punto de corte correspondiente a la menor probabilidad proporcionada por cada ventana.



Elección del número de transiciones a estudiar y puntos de corte

Eligiendo la opción de una ventana de 4 valores consecutivos, p3t, el punto de corte que se usaría para declarar una secuencia como anómala sería: 5.72902431135337e-06.

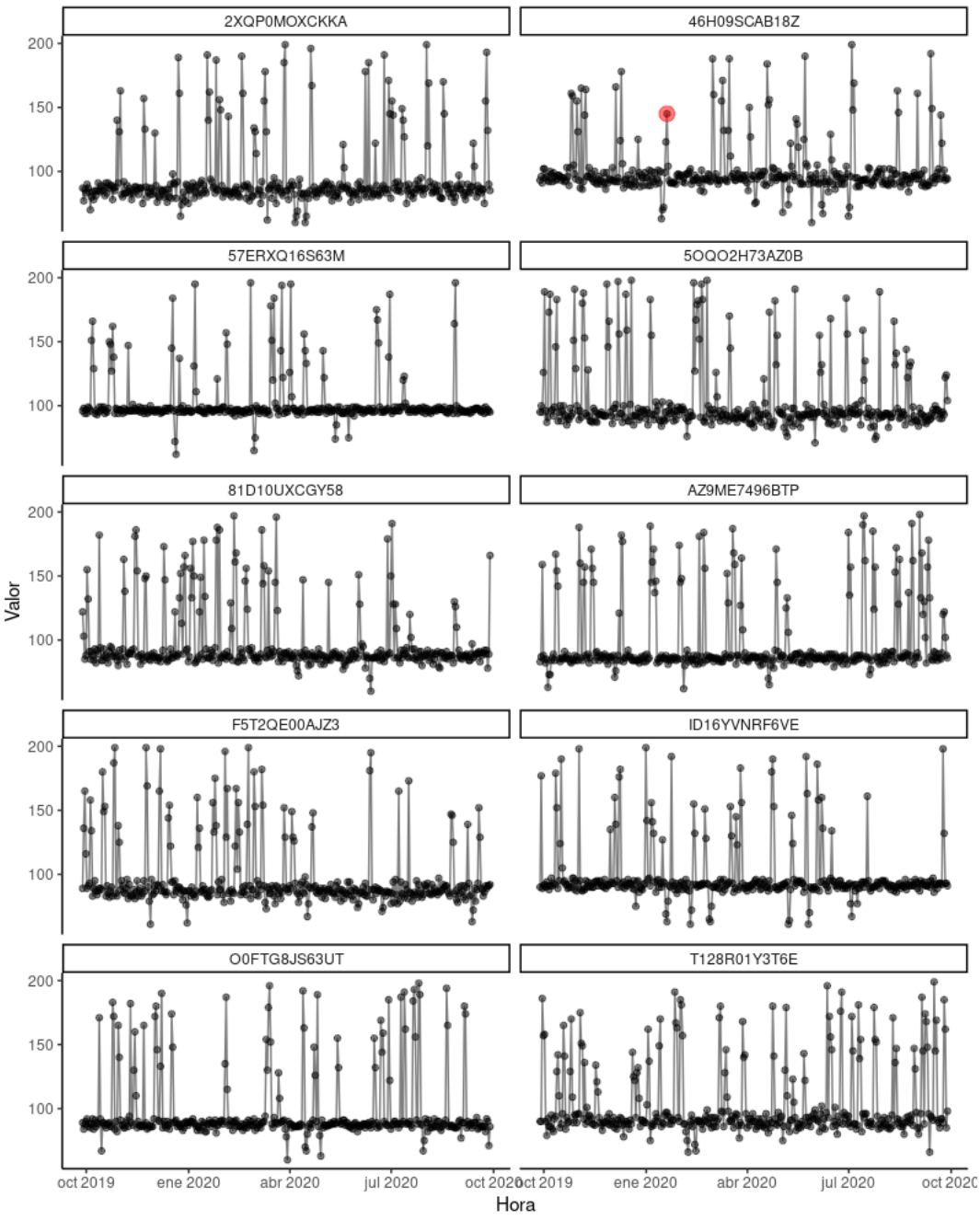
```
renderPlot({
dfFinal() %>% ggplot(aes(x=pElegida))+geom_histogram(color="navyblue",fill="darkred")+
  scale_x_log10()+
    geom_rug()+
    xlab(str_c("Probabilidad de ",input$numTransiciones, " transiciones"))+
    geom_vline(xintercept = pc(),color="red")+
    theme_minimal()
},width = 800,height = 300,res=100)
```



A continuación, se va a marcar con colores la probabilidad de las diferentes transiciones en cada punto.

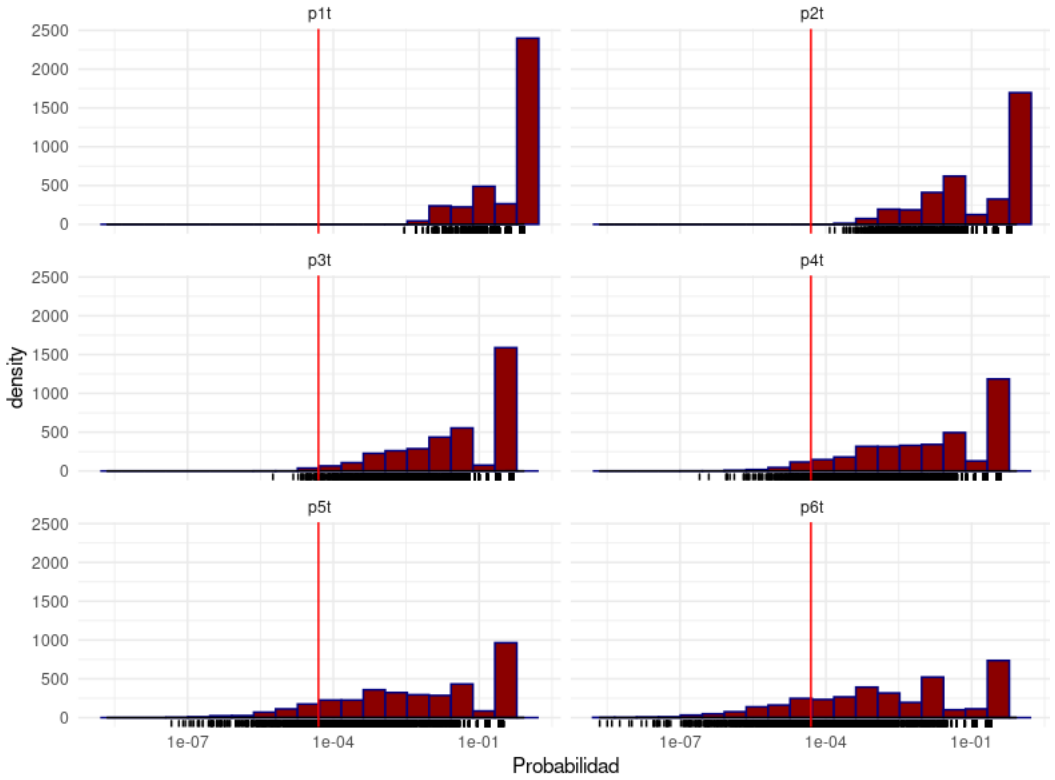
```
renderPlot({


  puntoCorte=pc()
ggplot(dfFinal(),aes(x=Hora,y=Valor))+
  geom_point(alpha=0.5)+
  geom_line(alpha=0.5)+
  geom_point(data=dfFinal() %>% filter(pElegida<=puntoCorte),
             aes(x=Hora,y=Valor),color="red",size=4,alpha=0.5)+
  facet_wrap(~Device,ncol = 2)+
      theme_classic()


},width = 800,height = 1000,res=100)
```

Como se ha seleccionado la ventana de 3 transiciones con menor probabilidad como punto de corte no es de extrañar que en la siguiente gráfica solo se pueda detectar un punto rojo como anómalo: aquel que ha originado el punto de corte establecido.

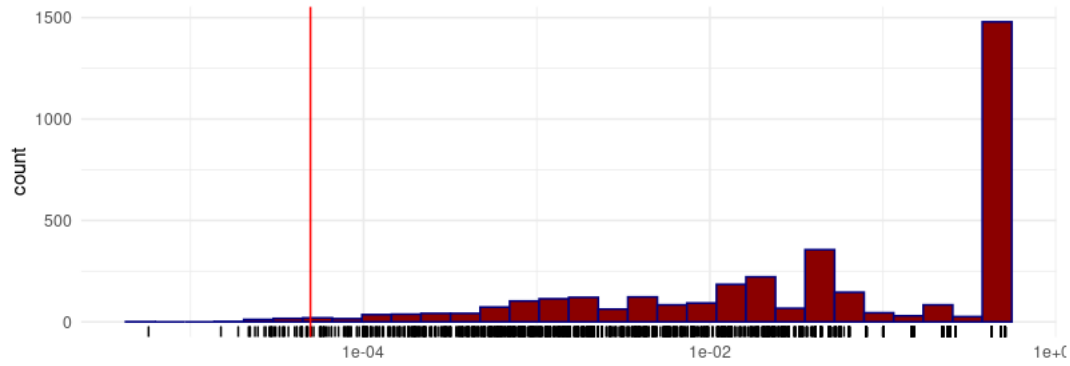


Análisis de parámetros para reducir falsos positivos

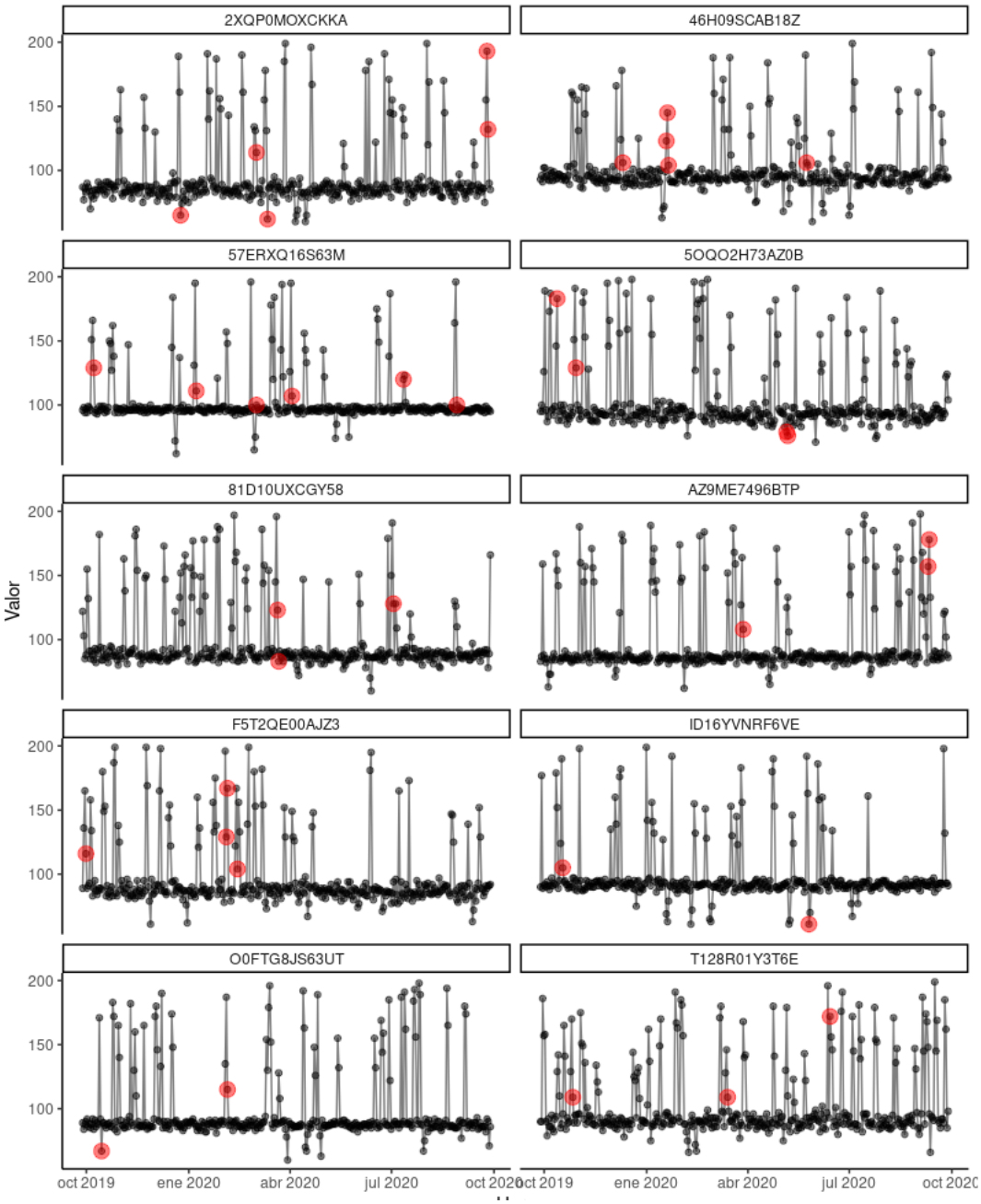
Estableciendo, por ejemplo, el punto de corte en el percentil 1, se pueden observar las siguientes gráficas, cada una en función de una ventana de transición. Recordemos que el punto de corte está marcado con una línea roja.



Se puede observar que parece una buena opción elegir la ventana en la que se estudian 4 estados consecutivo. O sea, 3 transiciones: p3t.



Al elegir de las transiciones anómalas el 1% de las más improbables se podrían detectar mejor las anomalías:



Un buen método para probar el programa sería, por ejemplo, si son 10 dispositivos en el "training.txt", entrenar con 6 de ellos y probar la ejecución con los 4 restantes.

Pruebas

Se ha creado una nueva clase, `Datos`, para crear los documentos de texto que comprobarán la correcta ejecución del programa.

Además, en la clase principal se han creado tres formas diferentes para ejecutar el programa. Serán descritas a continuación.

```java
public static void main(String[] args) {
    if(args.length == 0) {
        main1();
    }else {
        if(args[0].equals("prueba.txt")) {
            main1f(new File(args[0]));
        }else if(args[0].equals("test.txt")) {
            main2f(new File(args[0]));
        }else {
            main1f(new File(args[0]));
        }
    }
}
```

Ilustración 16. main()

1. **Main con medidas que se leen de teclado**

```java
// Para probar el programa con training.txt como entrenamiento
public static void main1() {
    // Apartado 1
    try {
        leerFichero();
    } catch (IOException e) {
        e.printStackTrace();
    }
    List<Integer> valores = Matriz.obtenerValores();
    int menor = Matriz.menor(valores);
    int mayor = Matriz.mayor(valores);

    Matriz.puntoCorte(menor, mayor, valores);
    // Matriz.matrixN2file(menor, mayor, valores);
    Matriz.matrixP2file(menor, mayor, valores);
    Matriz.PC2file(menor, mayor, valores);

    // Apartado 2
    try {
        Datos.datos();
        /* Para crear los datos que se introducirán por teclado
         * Son las mismas mediciones que training.txt */
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    try {
        DetectorAnomalias.detectarAnomalia(menor, mayor);
        /* Copiar de prueba.txt las mediciones y pegarlas en la terminal
         * insertando "FIN" al final
         * Si funciona correctamente devolverá únicamente la ventana que
         * originó el punto de corte como anomalía */
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

Ilustración 17. main1()

En la clase `Datos` está declarado el siguiente método:

```java
public static void datos() throws IOException {
    Writer wr = new FileWriter("prueba.txt");
    List<Integer> numeros = Matriz.obtenerValores();
    for(int i = 0; i < numeros.size(); i++) {
        int num = numeros.get(i);
        if(num != -1) { // Para obviar el cambio de máquina
            wr.write(String.valueOf(num) + "\n");
        }
    }
    wr.close();
}
```

Ilustración 18. Crear un fichero en el que se insertan las medidas del fichero de entrenamiento

2. Main para probar el programa con "training.txt" como entrenamiento y testeo

Lee el fichero "training.txt" y las medidas obtenidas se insertan en un fichero, que será el que se lea por la consola y dará como único punto anómalo la ventana que creó el punto de corte. En el fichero "README" muestra cómo probar el programa.

```java
public static void main1f(File file) {
    // Apartado 1
    try {
        leerFichero();
    } catch (IOException e) {
        e.printStackTrace();
    }

    List<Integer> valores = Matriz.obtenerValores();
    int menor = Matriz.menor(valores);
    int mayor = Matriz.mayor(valores);

    Matriz.puntoCorte(menor, mayor, valores);
    Matriz.matrixN2file(menor, mayor, valores);
    Matriz.matrixP2file(menor, mayor, valores);
    Matriz.PC2file(menor, mayor, valores);

    // Apartado 2
    try {
        Datos.datos();
        /* Para crear los datos. Son las mismas mediciones que training.txt */
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    try {
        List<Integer> contenido = leerMedidasFichero(file);
        DetectorAnomalias.detector(menor, mayor, contenido);
        /* Si funciona correctamente devolverá únicamente la ventana que
         * originó el punto de corte como anomalía */
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

Ilustración 19. main1f()

El resultado de la terminal es el siguiente:

```
Punto de corte: 5.729024311353374E-6
Anomalía en la ventana de mediciones: 72 - 99 - 123 - 145
```

```
Probabilidad: 5.729024311353374E-6
```

Comprobamos que funciona correctamente al dar el resultado esperado, coincidiendo este con el de R.

3. Main para probar con 6 dispositivos de "training.txt" el entrenamiento y con los 4 restantes el testing.

En la clase `Datos` están implementados los siguientes métodos:

- **Método de entrenamiento**

  Recibe por parámetro todos los valores de "training.txt" y devuelve las medidas de sus 6 primeros dispositivos, incluyendo el cambio de máquina con "-1". Estos servirán para realizar el entrenamiento en la clase `Matriz`. En mi caso, los 6 primeros dispostivos incluyen: T128R01Y3T6E, ID16YVNRF6VE, F5T2QE00AJZ3, 57ERXQ16S63M, 81D10UXCGY58 y 2XQP0MOXCKKA. Además, crea también una lista para agregar las medidas de los 4 dispositivos restantes, sin considerar el cambio de máquina, que se empleará para realizar el testeo.

```
/* Considerando que hay 10 dispositivos en training.txt, se emplearán los
 * 6 primeros para el aprendizaje y los 4 siguientes para el testeo */
public static List<Integer> entrenamiento(List<Integer> valores) throws IOException {
    List<Integer> entrenamiento = new ArrayList<>();
    List<Integer> testeo = new ArrayList<>();
    int cnt = 0;
    for(int i = 0; i < valores.size(); i++) {
        int num = valores.get(i);
        if(num == -1) {
            cnt++;
        }
        if(cnt < 6) {    // Valores a entrenar
            entrenamiento.add(num);
        }else { // Valores a testear
            if(num != -1) {
                testeo.add(num);
            }
        }
    }
    testeo(testeo);
    return entrenamiento;
}
```

Ilustración 20. Creación de valores de entrenamiento y con los que entrenará, para el main2()

- **Método de testeo**

  Crea el fichero "test.txt". En este se almacenarán las medidas de los 4 dispositivos de "training.txt" que no fueron empleados para el entrenamiento.

```
public static void testeo(List<Integer> testeo) throws IOException {
    Writer wr = new FileWriter("test.txt");
    for(int i = 0; i < testeo.size(); i++) {
        wr.write(String.valueOf(testeo.get(i)) + "\n");
    }
    wr.close();
}
```

Ilustración 21. Creación del fichero para el testing en el main2()

Por lo tanto, `main2` quedaría de la siguiente forma:

```
// Entrena con 6 dispositivo y testea con 4, del training.txt
public static void main2f(File file) {
    // Apartado 1
    try {
        leerFichero();
    } catch (IOException e) {
        e.printStackTrace();
    }

    List<Integer> vals = Matriz.obtenerValores();
    List<Integer> valores = new ArrayList<>();
    try {
        valores = Datos.entrenamiento(vals);    // Devuelve los datos a entrenar
        // Se crean los datos a testear
    } catch (IOException e1) {
        e1.printStackTrace();
    }
    int menor = Matriz.menor(valores);
    int mayor = Matriz.mayor(valores);

    Matriz.puntoCorte(menor, mayor, valores);
    Matriz.matrixN2file(menor, mayor, valores);
    Matriz.matrixP2file(menor, mayor, valores);
    Matriz.PC2file(menor, mayor, valores);

    // Apartado 2
    try {
        List<Integer> contenido = leerMedidasFichero(file);
        DetectorAnomalias.detector(menor, mayor, contenido);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

Ilustración 22. main2()

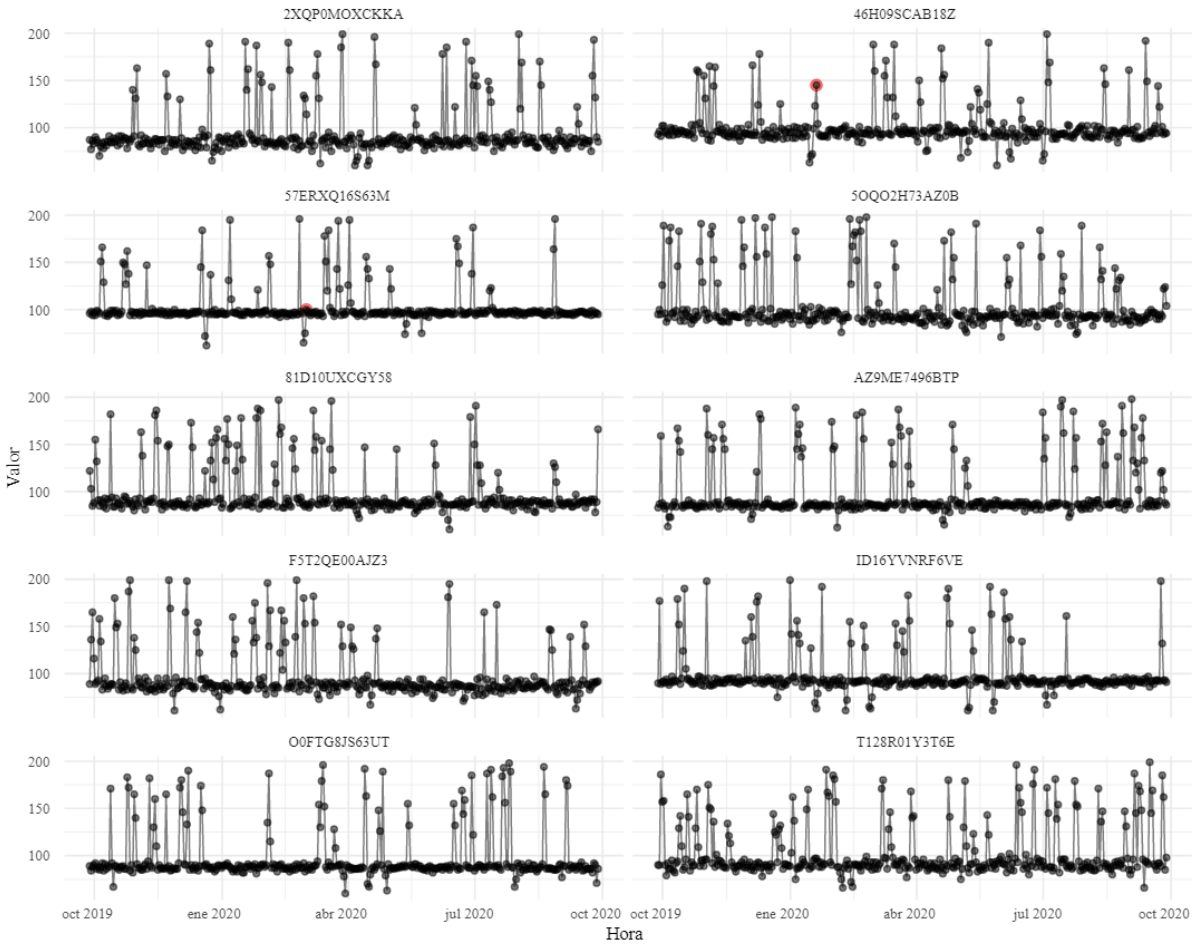
En el fichero "README" se muestra cómo probar el programa. El resultado de la terminal es el siguiente:

```
Punto de corte: 6.631585578954001E-6
Anomalía en la ventana de mediciones: 72 - 99 - 123 - 145
      Probabilidad: 2.7644813902024015E-6
```

Para comprobar que funciona bien, vemos si en R sale el mismo resultado:

```
renderTable({
  dfEtiquetado() %>% filter(pElegida<=pc()) %>% select(Device,Hora,Valor,tooltip,Entrenamiento)
})
```

Device	Hora	Valor	tooltip	Entrenamiento
57ERXQ16S63M	1583074389.00	100	96=>65=>75=>100 prob=6.631585578954e-06	TRUE
46H09SCAB18Z	1579446362.00	145	72=>99=>123=>145 prob=2.7644813902024e-06	FALSE



El programa funciona correctamente.