Mejora y actualización de las funcionalidades del paquete estadístico regioneR

Sergio Obon Temprano

MU Bioinformàtica y Bioestadística
Desarrollo de programas y aplicaciones

Nombre Tutor/a de TFM
Bernat Gel Moreno
Profesor/a responsable de la asignatura
Antoni Pérez Navarro

20/06/2023

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Copyright © 2023 Sergio Obon Temprano

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

© Sergio Obon Temprano
Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Mejora y actualización de las funcionalidades del paquete estadístico regioneR*
Nombre del autor:	*Sergio Obon Temprano*
Nombre del consultor/a:	*Bernat Gel Moreno*
Nombre del PRA:	*Antoni Pérez Navarro*
Fecha de entrega (mm/aaaa):	*06/2023*
Titulación o programa:	Máster en Bioinformática y Bioestadística
Área del Trabajo Final:	*Desarrollo de programas y aplicaciones*
Idioma del trabajo:	*Castellano*
Palabras clave	*Permutation test, R functions, regioneR*

Resumen del Trabajo

Los test de permutación son una herramienta matemática muy útil para temas bioinformáticos. El paquete regioneR, a partir de test de permutación, establece relaciones entre conjuntos de regiones genómicas y posibles características de estas. El objetivo de este trabajo era desarrollar dos funciones que pudieran integrarse a este paquete. Para ello se trabajó en R, un lenguaje de programación gratuito y de código abierto. Por un lado, se desarrolló la función *clusteringScore* que, dado un conjunto de regiones, devuelve un valor que indica cómo de clusterizadas están. Por otro, se desarrolló la función *similarRegions* que, dado un conjunto de regiones y una función de evaluación, devuelve otra región similar según dicha función de evaluación. Ambas funciones fueron testeadas con conjuntos de regiones aleatorias y con datos biológicos reales. Los resultados mostraron que la función *clusteringScore* devolvía un p-valor que permitía distinguir entre conjuntos aleatorios y conjuntos altamente clusterizados y que la función *similarRegions* nos permitía obtener un conjunto similar al conjunto dado según una función de evaluación con un parámetro de tolerancia ajustable por el usuario. Las funciones desarrolladas complementan las funcionalidades del paquete regioneR y tienen un gran número de aplicaciones potenciales que pueden contribuir al estudio de asociaciones genómicas y al desarrollo de técnicas bioinformáticas y bioestadísticas.

Abstract

Permutation tests are a very useful mathematical tool for bioinformatics. The regioneR package, based on permutation tests, establishes relationships between sets of genomic regions and possible characteristics of these regions. The objective of this work was to develop two functions that could be integrated into this package. To this end, we worked in R, a free and open-source programming language. On the one hand, we developed the function clusteringScore which, given a set of regions, returns a value indicating how clustered they are. On the other hand, we developed the function similarRegions which, given a set of regions and an evaluation function, returns another similar region according to that evaluation function. Both functions were tested with random sets of regions and with real

biological data. The results showed that the clusteringScore function returned a p-value that allowed us to distinguish between random sets and highly clustered sets and that the similarRegions function allowed us to obtain a set similar to the original one according to an evaluation function with a customizable tolerance parameter. The developed functions complement the functionalities of the regioneR package and have a large number of potential applications that can contribute to the study of genomic associations and to the development of bioinformatics and biostatistical techniques.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1. Contexto y justificación del trabajo

Los avances de las tecnologías de secuenciación del ADN y la capacidad computacional para analizar un gran volumen de datos han proporcionado nuevas metas a la bioinformática, haciendo posible el mapeado completo del genoma humano. Estos avances han ayudado a determinar las funciones de genes y proteínas y a entender las causas y el desarrollo de muchas enfermedades, abriendo un gran abanico de posibilidades a la hora de aplicar nuevos tratamientos [1].

La bioinformática, gracias a su capacidad para analizar gran volumen de datos, juega un papel esencial en la interpretación de los datos genómicos, transcriptómicos y proteómicos, lo que ha desembocado en la posibilidad de llevar a cabo una medicina más personalizada [2].

Este trabajo es un proyecto de bioinformática dentro de la rama de desarrollo de programas y aplicaciones. Uno de los lenguajes de programación más utilizados en el campo de la bioinformática es R. R es una potente herramienta matemática que es de utilidad tanto para usuarios a un nivel básico de análisis de datos, como para avanzados proyectos de investigación. R contiene diferentes paquetes de funciones para múltiples aplicaciones. Bioconductor es un repositorio de paquetes de R, que nació como un proyecto de desarrollo de software gratuito para facilitar el análisis de datos de ensayos biológicos [3]. Así, R y Bioconductor son de gran utilidad para el análisis de datos ómicos y contienen distintas funcionalidades orientadas a esta rama.

En el campo de la bioinformática se trabaja con un gran volumen de datos ómicos, por lo que la búsqueda de información y el análisis estadístico pueden suponer un elevado coste computacional. Por ello, hay técnicas y herramientas matemáticas que pueden ayudar a reducir este coste y facilitar el manejo de datos. Un test de permutación (“permutation test”) es un test estadístico que compara dos o más muestras para comprobar si lo que observamos se debe al azar. La hipótesis nula no asume ninguna distribución específica de los datos, sino que la establece a partir de permutaciones. Los test de permutación se pueden entender como una forma de remuestreo, dado que la hipótesis nula de un test estadístico se obtiene a partir de permutaciones de los datos originales [4].

En 2016 se publicó regioneR, un paquete de R/Bioconductor que, a partir de test de permutación, establece relaciones entre conjuntos de regiones genómicas y posibles características de estas. Este paquete ofrece distintas funcionalidades en relación con la aleatorización y estrategias de evaluación de las distintas regiones [5].

En este contexto, tras hablar con los autores del paquete regioneR y discutir posibles necesidades y opciones de mejora, surgió la idea de este trabajo. En él, se pretende ampliar las funcionalidades de regioneR, añadiendo dos funciones, una que evalúe la clusterización de un conjunto de regiones y otra que devuelva un conjunto de regiones similar a una región dada, según la evaluación de una determinada función. Cuantificar el grado de clusterización de un conjunto de regiones es algo que todavía no está implementado, pero puede ser de gran utilidad para saber cómo se distribuye a lo largo del genoma un determinado conjunto de datos biológicos. Por otro lado, la segunda función nos permitiría, a partir de un conjunto de regiones con una determinada característica (por ejemplo, el porcentaje de bases con una determinada secuencia), crear artificialmente otros conjuntos de regiones que mantengan dicha característica.

1.2. Objetivos del Trabajo

RegioneR es un paquete de R que implementa una serie de funciones, basadas en test de permutación, para trabajar con regiones del genoma. RegioneR tiene diversas estrategias de aleatorización y evaluación y ofrece al usuario la posibilidad de utilizar funciones con parámetros personalizables [5]. Hablando con los creadores de regioneR, me plantearon la posibilidad de ampliar las opciones del paquete, añadiendo nuevas funciones que cubrieran algunas de las necesidades principales reclamadas por los usuarios. Por ello, los objetivos de este trabajo son:

1) Desarrollar una función que, dado un conjunto de regiones del genoma, devuelva un índice cuantitativo que indique cómo de clusterizadas están.
2) Desarrollar una función que, dado un conjunto de regiones y una función de evaluación que devuelve un valor numérico, genere otro conjunto de regiones con una evaluación similar de acuerdo a la función dada.
3) Elaborar funciones de evaluación, a modo de ejemplo, que, además, sirvan para testear la función anterior. Se diseñarán dos funciones: una de ellas evaluará el porcentaje de purinas en una región y la otra, dada una secuencia, dará el porcentaje de bases en los que se encuentra dicha secuencia.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Este trabajo se basa en la ampliación de las funcionalidades de un paquete de R. Uno de los puntos fuertes de R es que es un lenguaje gratuito y de código abierto. Esta gratuidad hace que cualquier persona pueda tener acceso a él sin necesidad de tener que pagar una licencia. El código abierto permite que, una vez diseñada una función o un paquete, no solo cualquier usuario tenga acceso a dicha función, sino que también tiene acceso a su código, permitiendo así un avance global del conocimiento.

1.4. Enfoque y método seguido

Como se ha indicado anteriormente, el principal objetivo de este trabajo es ampliar las funcionalidades del paquete de R regioneR, con el diseño de dos nuevas funciones que se incorporarán a dicho paquete.

Para ello, en ambos casos se han seguido los mismos pasos:

- Diseño de la función: elaboración del código de la función que realice la funcionalidad deseada.
- Implementación y validación de la función: realización de test que permitan corroborar que, efectivamente, la función diseñada realiza esta función, comprobando los resultados obtenidos en cada caso y aplicando las modificaciones y mejoras necesarias.
- Integración de la función en el paquete regioneR: una vez validada la función, se integra al paquete, siendo una funcionalidad más de este.

1.5. Planificación del Trabajo

1.5.1. Tareas

- Familiarizarse con el paquete regioneR (1 semana)
- Diseñar una función de clusterización (4 semanas)
    - Diseñar la función que permita la clusterización (1 semana)
    - Implementar y probar la función y realizar posibles correcciones para validarla (2 semanas)
    - Integrar la función en el paquete regioneR (1 semana)
- Establecer un sistema de randomización a partir de unas premisas aportadas por el usuario (5 semanas)

- Diseñar la función de randomización (1 semana)
- Implementar y probar la función y realizar posibles correcciones para validarla (2 semanas)
- Diseñar las funciones de ejemplo (1 semana)
- Integrar la función en el paquete regioneR (1 semana)

- Escribir la memoria (3 semanas)
- Preparar la presentación (2 semanas)

1.5.2. Calendario

En la Figura 1, se muestra el diagrama de Gantt con la planificación de las tareas y el calendario.

Figura 1: diagrama de Gantt

1.5.3. Hitos

Los hitos que se plantean para las distintas etapas del trabajo son:

PEC 2:

- Haber diseñado una función que permita la clusterización y funcione con datos simulados.
- Tener una función de clusterización validada y testeada con datos reales.

PEC 3:

- Haber diseñado un sistema de randomización que funcione con datos simulados.
- Tener un sistema de randomización validado y testeado con datos reales.
- Haber diseñado las dos funciones ejemplo.

PEC 4:

- Haber integrado la función de clusterización en el paquete regioneR.

- Haber integrado la función de randomización en el paquete regioneR.
- Tener la memoria completada.
- Tener elaborada la presentación.

1.6. Breve sumario de productos obtenidos

En este trabajo se han diseñado e integrado a regioneR dos funciones. La primera de ellas, *clusteringScore*, dado un conjunto de regiones, devuelve un índice que indica cómo de agrupadas o "clusterizadas" están.

La segunda función que se ha integrado es *similarRegions* que, dado un conjunto de regiones y una función de evaluación, devuelve un conjunto de regiones similares de acuerdo a la función de evaluación dada.

1.7. Breve descripción de los otros capítulos de la memoria

A continuación, se describe la estructura de la memoria de este trabajo. Este primer capítulo corresponde a la Introducción, con el contexto, justificación, objetivos y planificación del trabajo. En el capítulo 2, se presentará una revisión del estado del arte. Se hablará de los test de permutación, de Bioconductor y de regioneR, explicando algunas de las funciones del paquete que guardan relación con las desarrolladas en este trabajo.

En el capítulo 3, correspondiente a los materiales y métodos, se describirán detalladamente las funciones y su utilidad y se hablará del proceso seguido para su desarrollo.

En el capítulo 4, se presentarán los resultados, incluyendo el código de las distintas funciones desarrolladas y la demostración de su utilidad con algunos ejemplos para cada caso.

En el capítulo 5, se presentarán las conclusiones obtenidas y se tratará sobre posibles trabajos futuros.

Finalmente, las secciones finales contendrán el glosario y la lista de referencias utilizadas.

2. Estado del arte

El gran volumen de datos que se debe analizar en la bioinformática ha obligado a desarrollar distintas técnicas para poder gestionarlos. Una de ellas son los test de permutación ("permutation test" en inglés). A diferencia de los test estadísticos clásicos, los test de permutación no buscan una correlación entre variables, sino que, a partir de los datos analizados, estiman directamente la significación estadística [6]. Es decir, dado un conjunto de datos, nos indica cómo de probable es haberlos obtenido aleatoriamente.

Este tipo de test ha sido utilizado para evitar las condiciones excesivamente restrictivas de la corrección de Bonferroni. Esta corrección divide el nivel de significación entre el número de comparaciones realizadas, lo que provoca que con un gran volumen de comparaciones sea excesivamente restrictiva [7].

Los test de permutación han sido muy utilizados en distintas aplicaciones médicas, como por ejemplo para el análisis de imágenes de resonancia magnética [8] o para identificar los genes asociados con la relación entre el tabaco y la esquizofrenia [9].

Bioconductor es un proyecto gratuito, de código abierto, basado en el lenguaje de programación R, que tiene como objetivo facilitar el análisis de datos de ensayos biológicos [3]. Bioconductor contiene más de 2000 paquetes, de software, de anotación de datos, de datos experimentales o de workflow [10]. Uno de los paquetes de Bioconductor es regioneR. En regioneR han desarrollado una serie de funciones que, empleando test de permutación, establecen relaciones entre conjuntos de regiones genómicas y posibles características de estas [5].

Una de las funciones de regioneR más relevantes para este trabajo es *permTest* [11]. Esta función realiza un test de permutación para comprobar si existe una relación estadísticamente significativa entre el conjunto de regiones dado y alguna característica según una función de evaluación. La función recibe distintos parámetros, principalmente el conjunto de datos a evaluar, la función de randomización con la que se van a obtener los conjuntos aleatorios y la función de evaluación, mediante la cual se va a buscar la relación entre los conjuntos.

La función *permTest* tiene múltiples aplicaciones y se ha utilizado para testear la significancia de los análisis en, por ejemplo, estudios de genes relacionados con la calidad de la carne [12], comparaciones epigenómicas del estado de la cromatina en distintas especies de mamíferos [13], o para establecer asociaciones entre mutaciones y la leucemia mieloide aguda [14].

Otra de las funciones de regioneR especialmente relevante para este trabajo es *createRandomRegions* [15]. Esta función se utiliza para crear regiones aleatorias. Recibe como parámetros el número de regiones a crear, la media de las regiones creadas, la desviación estándar del tamaño de las regiones, el genoma y si se desea permitir que las regiones solapen entre ellas.

CreateRandomRegions se ha utilizado en distintos trabajos para crear regiones aleatorias y compararlas con las observaciones del estudio. Por ejemplo, Marchal et al. la utilizaron en su estudio de la regulación genómica de la retina humana [16].

RegioneR tiene múltiples funciones, pero no dispone todavía de dos funcionalidades que irían muy relacionadas a las características de este paquete. Por un lado, dado un conjunto de regiones, podríamos preguntarnos cómo de clusterizadas están, es decir, cómo de agrupadas están dichas regiones. Por ello, en este trabajo, se pretende crear una función que devuelva un valor que sirva como índice de clusterización. Por otro lado, regioneR dispone de una función para la creación de regiones aleatorias (*createRandomRegions*), pero sería interesante no solo poder generar artificialmente regiones aleatorias sin ninguna condición, sino regiones que cumplan con un determinado criterio, siendo similares a otro conjunto dado. Por ello, en este trabajo también se pretende crear una función que, dado un conjunto de regiones, devuelva un conjunto de regiones aleatorias similares al conjunto dado. Para dicho propósito, se utilizará una función de evaluación y se buscará que dé un resultado similar para ambos conjuntos.

3. Materiales y métodos

El paquete regioneR es un paquete desarrollado mediante el lenguaje de programación R, un lenguaje gratuito y de código abierto. Para este trabajo se ha utilizado R Studio [17], un entorno de desarrollo integrado para dicho lenguaje de programación. Este entorno facilita la escritura del código y permite ejecutarlo desde el propio entorno.

El objetivo principal de este trabajo era desarrollar dos funciones e integrarlas al paquete regioneR. La primera de ellas, denominada *clusteringScore*, dado un conjunto de regiones, devuelve un índice indicando cómo de agrupadas (o "clusterizadas") están. La segunda, llamada *similarRegions*, dado un conjunto de regiones, devuelve un conjunto de regiones similares de acuerdo con una función de evaluación.

Para ambas funciones se siguió una metodología similar:

- Diseño de la función
- Testeo de la función con ejemplos prácticos
- Integración de la función en el paquete regioneR

A continuación, en los apartados 3.1 y 3.2 se explicará el diseño y testeo de las funciones *clusteringScore* y *similarRegions*, respectivamente; mientras que en el 3.3 se explicará cómo se integraron ambas funciones en el paquete regioneR.

3.1. ClusteringScore

La idea de esta función surge de la necesidad de encontrar una medida cuantitativa que, dado un conjunto de regiones, nos indique cómo de agrupadas están.

Para el primer prototipo, primeramente, se diseñó una función que denominamos *calculateDistances*, que, dado un conjunto de regiones, devolvía una matriz con las distancias entre todas las regiones dos a dos. Seguidamente, se seleccionaban los *n* valores más bajos y se calculaba la media, obteniendo un valor para cada región en función de los *n* vecinos más próximos. Se devolvía la mediana de estos valores, y se obtenía así un valor de clusterización para cada conjunto de regiones. Finalmente, se utilizaba la función *permTest* [11] para obtener un p-valor asociado al nivel de clusterización.

La función se testeó, primeramente, calculando el nivel de clusterización de conjuntos de prueba aleatoriamente generados con *createRandomRegions* [15]. Estos conjuntos tenían 100 regiones con una longitud media de 1000 bases y una desviación estándar de 200.

Seguidamente, se testeó calculando el nivel de clusterización de datos biológicos reales de mutaciones relacionadas con el cáncer de mama. El primer conjunto se componía de 118 regiones, mientras que el segundo se componía de 114.

En el caso de conjuntos aleatorios, el índice de clusterización podía resultar cualquier valor, pero sirvió para comprobar que el algoritmo funcionaba. Con los conjuntos de datos reales, se observó que había una tendencia a obtener p-valores pequeños, ya que tienden a estar clusterizados.

Para tratar de evitarlo, se diseñó un algoritmo que iba añadiendo al conjunto original conjuntos aleatoriamente generados (de 10% en 10% del tamaño original) y realizando *permTest* hasta que se obtenía un conjunto con un p-valor no significativo (el determinado por el usuario, >0.05 por defecto). El porcentaje de conjuntos añadidos era lo que se consideraba el índice de clusterización, ya que cuanto más clusterizado esté el conjunto original, mayor cantidad de conjuntos habrá que añadir para que el p-valor llegue a ser no significativo. Sin embargo, este método se descartó por dos razones. Primero, al añadir conjuntos aleatorios, había mucha variabilidad en el resultado y, además, la realización de múltiples *permTest* hacía que el coste computacional fuera muy alto.

Por tanto, se decidió recuperar el primer algoritmo y se realizaron algunas modificaciones. El algoritmo final de la función es el siguiente:

1) Recibe un conjunto de regiones. Sea *r* el número de regiones.
2) Calcula las distancias entre todas las regiones dos a dos.
3) Para cada región, selecciona las *n* distancias $d_1,\ldots,d_n$ menores.
4) Calcula las medias de $d_1,\ldots d_n$ para cada región. Sean $m_1,\ldots,m_r$ dichas medias.
5) Sea *s* el número de valores m_i iguales a infinito. Se eliminan los *s* valores infinitos y los *s* valores más pequeños. Sean $m_{s+1},\ldots,m_{r-s}$ las medias que se conservan.
6) Calcula la media M de $m_{s+1},\ldots m_{r-s.}$
7) Mediante la función *permTest* de regioneR [18] se calcula un p-valor que indica la probabilidad de haber obtenido M de manera aleatoria.

Cuando se obtiene el valor para cada región (paso 3), hay algunos valores que son infinito y, por tanto, no se puede realizar la media directamente. La principal diferencia entre el algoritmo final

y el primer prototipo es la manera de tratar dichos valores infinito. En el primer prototipo, para intentar evitar los valores infinito, se consideraba la mediana. En el nuevo algoritmo, si hay *s* valores que son infinito, se descartan estos y los *s* valores más pequeños y se realiza la media de los valores restantes (pasos 5 y 6). Al final, a partir de este valor obtenido, se realiza un test de permutación que devuelve un p-valor y se considera como índice de clusterización, aun sabiendo que puede haber una tendencia a obtener valores bajos.

El primer ejemplo con el que se testeó la función fue con un conjunto de regiones creado aleatoriamente. Se consideró un conjunto de 100 regiones, de longitud media 1000 bases y desviación estándar de 200. Al aplicar la función *clusteringScore*, se consideraron 1000 iteraciones y se tuvieron en cuenta entre 1 y 3 vecinos. Se eligieron estos números porque, siendo *n* el número de vecinos, en caso de que haya menos de *n+1* regiones en un determinado cromosoma, se les asignará un valor infinito a todas ellas. Un número muy grande de valores infinito dificulta la obtención del p-valor.

Seguidamente, se testeó la función con unos datos biológicos sobre mutaciones en el cáncer de mama. El primer conjunto de regiones se componía de 118 muestras, mientras que el segundo se componía de 114. Se aplicó la función a ambos conjuntos, pasando como parámetros 1000 permutaciones para el test y 3 vecinos para calcular el valor de la función.

3.2. SimilarRegions

Esta función ofrece la posibilidad de, a partir de un conjunto de regiones y una función de evaluación dada, obtener un conjunto de regiones similares según dicha función de evaluación, aplicando un factor de tolerancia que también puede controlar el usuario.

El algoritmo que sigue la función *similarRegions* es el siguiente:

1) La función recibe un conjunto de regiones *A*, un factor de tolerancia *k* y una función de evaluación *f*, la cual devuelve un valor numérico para cada región.
2) Se evalúa cada región de *A* según *f* y se hace la media *M* de los valores devueltos.
3) Se genera un conjunto aleatorio *B* con el mismo número de regiones, del mismo tamaño y de la misma desviación estándar que *A*.
4) Se evalúa cada región de *B* según *f* y se hace la media *M'* de los valores devueltos.
5) Si la media M' está en el intervalo (*M-k,M+k*), se devuelve dicho conjunto y se acaba. Si no, se procede a los pasos siguientes.

6) Las regiones de *B* que están en el intervalo (*M-k,M+k*) se seleccionan y se descartan el resto.
7) Se completa el conjunto *B* con regiones aleatorias y se repite el procedimiento (pasos 4, 5 y 6). De este modo, se itera hasta que M' esté en (M-k,M+k), momento en el que se devuelve el conjunto *B* y finaliza el procedimiento.

Primeramente, se evalúa el conjunto de regiones dado, *A*, mediante la función de evaluación introducida como parámetro. Se calcula la media de estas evaluaciones y se obtiene un valor *M* que será al que trataremos de aproximar las evaluaciones del nuevo conjunto. Además, se calcula la media y la desviación estándar del tamaño de las regiones del conjunto dado.

Seguidamente, se crea un conjunto de regiones aleatorias, *B*, mediante la función *createRandomRegions* [15], de número de regiones, media y desviación estándar igual que la región *A*. Se evalúa cada una de estas regiones mediante la función de evaluación, se calcula la media y se comprueba si esta está cerca de *M*, según un factor de tolerancia *k*. Si lo está, se devuelve ese conjunto y, en caso contrario, se seleccionan las regiones de *B* que están cerca de M y el resto de las regiones se descartan. En este caso, el conjunto *B* se completa con regiones aleatorias y se repite el mismo procedimiento hasta que la media de las evaluaciones de las regiones de *B* esté suficientemente cerca de *M* o se alcance un número límite de iteraciones, que es otro parámetro controlable por el usuario.

Para poder probar esta función, se diseñaron dos funciones de evaluación a modo de ejemplo. Por un lado, una función que, dada una región, devuelve el porcentaje de purinas y, por otro, una función que, dada una región y una secuencia (por ejemplo, "AG"), devuelve el porcentaje de bases que ocupa la secuencia dada.

El primer testeo de la función fue para obtener un conjunto de regiones con un porcentaje de purinas similar a un conjunto inicial creado aleatoriamente. El conjunto inicial tenía 100 regiones, de longitud media 1000 bases y desviación estándar 200. Utilizando este conjunto como input, se ejecutó la función *similarRegions*, pasando como función de evaluación *purines_percentages* y distintos valores de tolerancia, *k*.
A continuación, testeamos la función *similarRegions* con un conjunto de 100 regiones aleatorias, creado con *createRandomRegions*, de media 1000 bases y desviación estándar 200, pasando como función de evaluación *sequence_percentage* y como secuencia *CAG*.
Finalmente, probamos la función para conjuntos de promotores. Los promotores son regiones de ADN que se encuentran justo antes de iniciar la transcripción de un gen [19]. Los promotores

tienen una concentración de secuencias GC mayor a la habitual [20]. Mediante la función *Promoters* del paquete BiSeq [21] descargamos los promotores. Se descargó en un elemento GRanges de 38294 elementos. Para realizar el ejemplo, se tomaron los 100 primeros, y se aplicó la función *similarRegions*, pasando como función de evaluación *sequence_percentage*, como secuencia "GC" y como *k* se probaron los valores 0.05, 0.01, 0.005.

3.3. Integración en regioneR

Para integrar las funciones en el paquete regioneR se documentaron las funciones, incluyendo una cabecera con la descripción, la manera en que se deben utilizar los parámetros y otra información relevante que se proporciona al usuario.

Una vez las funciones se validaron completamente, desde el GitHub del paquete regioneR, se creó un "Fork" donde se actualizó el paquete, añadiendo los ficheros .Rd en la carpeta *man* del GitHub de regioneR y sustituyendo el fichero *NAMESPACE*.

Seguidamente, desde el GitHub se clica en "Pull request" y se selecciona la opción "create a pull request" para solicitar la integración de las funciones en el paquete.

4. Resultados

4.1. Función *clusteringScore*

El principal objetivo era diseñar una función que, dado un conjunto de regiones, devolviera un valor que indicase cómo de agrupadas estaban dichas regiones. Finalmente, después de los distintos testeos se obtuvo una función cuyo código en R es el siguiente:

```
clusteringScore<-function(A,nt=100,neighbors=3,...){
  if(isEmpty(A)){
    stop("The input is empty")
  }
  # We define a function that, given a set of regions, returns
  # the distance matrix between them
  calculateDistances<-function(A,...){
    # We define a vector with length the number of regions given,
    # where each element is the corresponding chromosome number
    chromosomes<-vector(length=length(A))
    chromosomes<-as.character(seqnames(A))

    distances<-matrix(nrow=length(A),ncol=length(A))
    starts<-start(A)
    lengths<-width(A)
    for (i in 1:length(A)){
      # If two regions are in differents chromosomes, the distance is infinite
      # If twwo regions overlap, the distance is 0
      # In another case, the distance is the number of positions
      # separating the two regions
      for (j in 1:length(A)){
        if(chromosomes[i]!=chromosomes[j]){
          distances[i,j]=Inf
        }
        else{
          start_i<-starts[i]
          end_i<-starts[i]+lengths[i]+1
          start_j<-starts[j]
          end_j<-starts[j]+lengths[j]+1
          if(start_i<=start_j){
            if(end_i>=start_j){
              distances[i,j]<-0
            }
            else{
              distances[i,j]<-start_j-end_i
            }
          }
          else{
            if(end_j>=start_i){
              distances[i,j]<-0
            }
            else{
              distances[i,j]<-start_i-end_j
            }
          }
        }
      }
    }
    return(distances)
  }
  # We define a function that, given a matrix of distances, returns
  # the mean of each row
```

```
  calculateMeans<-function(distances,n,...){
    means<-vector(length=nrow(distances))
    for (i in 1:nrow(distances)){
      dist<-distances[i,-i]
      dist<-sort(dist)
      means[i]<-mean(dist[1:n])
    }
    return(means)
  }
  # We define a region that returns the mean of the ditances to a
  # given number of neighbors
  cluster_mean<-function(A,neighbors=3,...){
    d<-calculateDistances(A)
    x<-calculateMeans(d,neighbors)
    x<-sort(x)
    # Infinite values will not be considered
    # k smallest values will be eliminated,
    # where k is the number of infinite values
    ind_inf<-which(x==Inf)
    num_inf<-length(ind_inf)
    if(num_inf>=(length(x)/2)){
      return(Inf)
    }
    else{
      x<-x[-c(1:num_inf,ind_inf)]
      return(mean(x))
    }
  }
  # We calculate the p-value, using permTest function
  p<-permTest(A,ntimes=nt,randomize.function=randomizeRegions,evaluate.function=clust
er_mean,neighbors=neighbors)
  return (p)
}
```

4.2 Ejemplos de aplicación de *clusteringScore*

El primer ejemplo con el que se testeó la función fue con un conjunto de regiones creado aleatoriamente y se varió el número de vecinos a tener en cuenta. Los resultados de los índices de clusterización obtenidos para conjuntos de regiones aleatorias se presentan en la Tabla 1 y se grafican en las Figuras 2, 3 y 4.

Tabla 1: resultados de clusterización de conjunto aleatorio

Número de vecinos	p-valor	Evaluación del conjunto
1	0.27	$1.78 \cdot 10^7$
2	0.49	$2.97 \cdot 10^7$
3	0.37	$4.52 \cdot 10^7$

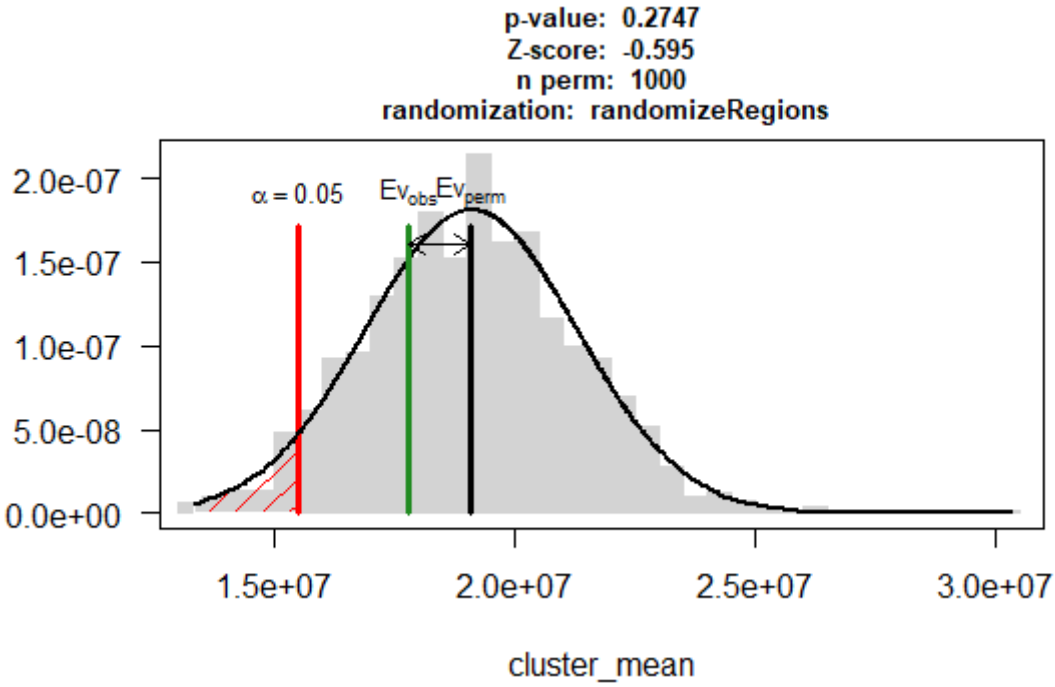

Figura 2: histograma resultante del permTest para un conjunto de regiones aleatorias y 1 vecino

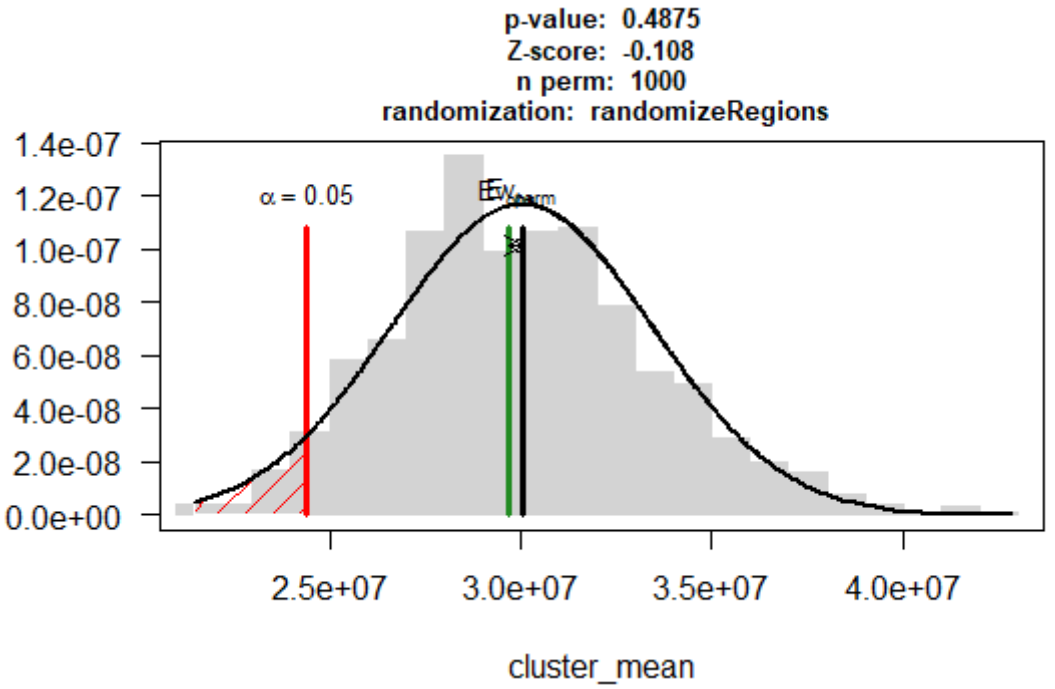

Figura 3: histograma resultante del permTest para un conjunto de regiones aleatorias y 2 vecinos

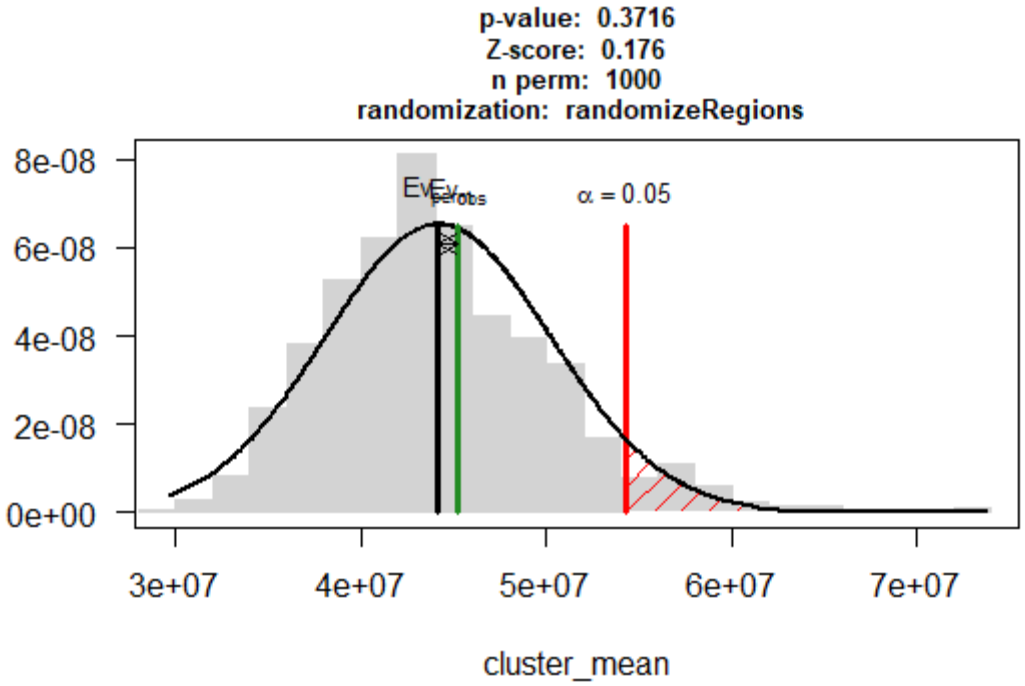

Figura 4: histograma resultante del permTest para un conjunto de regiones aleatorias y 3 vecinos

En las figuras anteriores, se observa en la zona sombreada el histograma correspondiente a las evaluaciones realizadas por *permTest*. La línea verde indica dónde está nuestra muestra, mientras que la línea roja indica dónde se encuentra el nivel de significación. Al tratarse de un conjunto de regiones creado de forma aleatoria, es normal que nuestra muestra se encuentre lejos de ser significativa.

Seguidamente, se testeó la función con unos datos biológicos sobre mutaciones en el cáncer de mama, que ya se intuía que estarían fuertemente clusterizados, como se observa al graficar el histograma sobre las distribuciones de los datos en los distintos cromosomas en las figuras siguientes. En la figura de una muestra de conjuntos aleatorios, se observa una distribución más homogénea, mientras que, en las muestras sobre las mutaciones en el cáncer de mama, las muestras están más concentradas en algunos cromosomas.

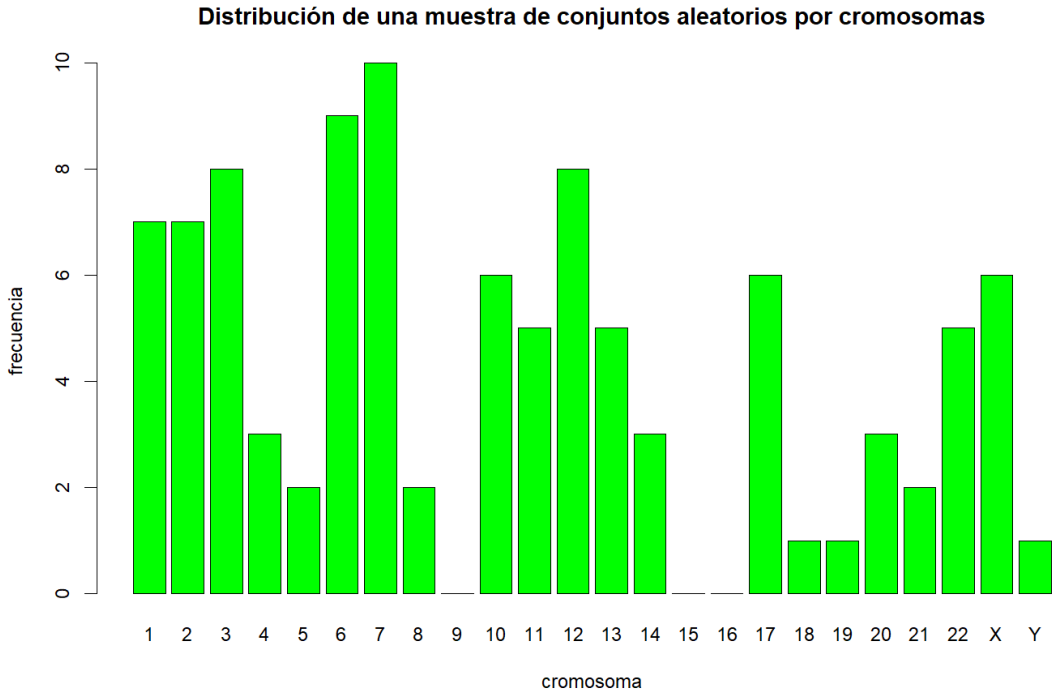

Figura 5: distribución de una muestra de conjuntos aleatorios por cromosomas

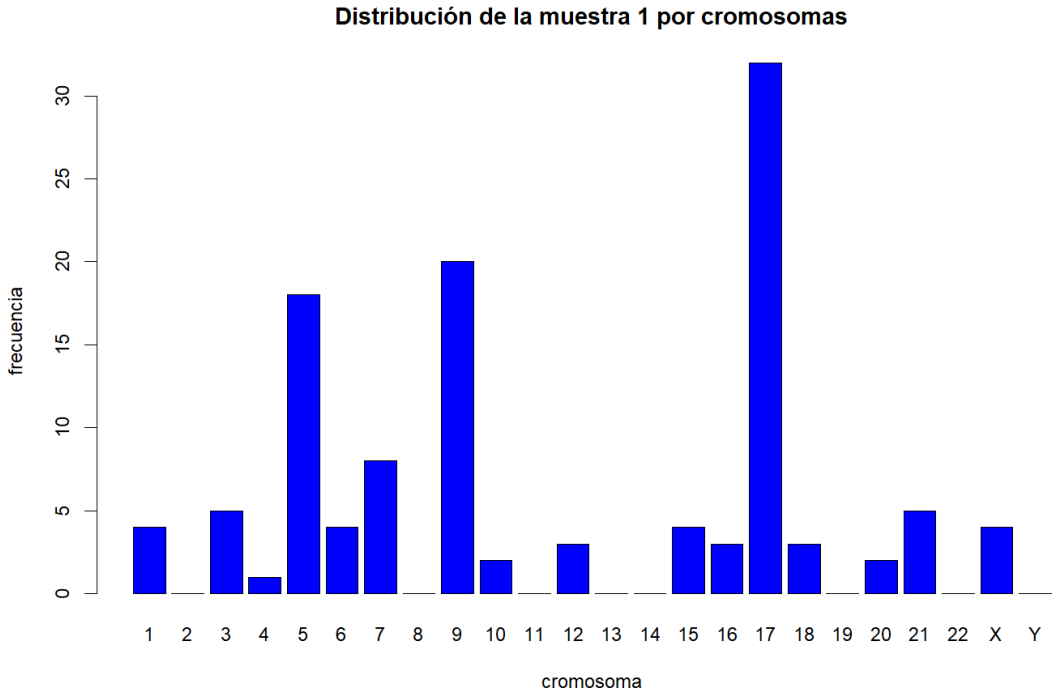

Figura 6: distribución de la muestra 1 por cromosomas

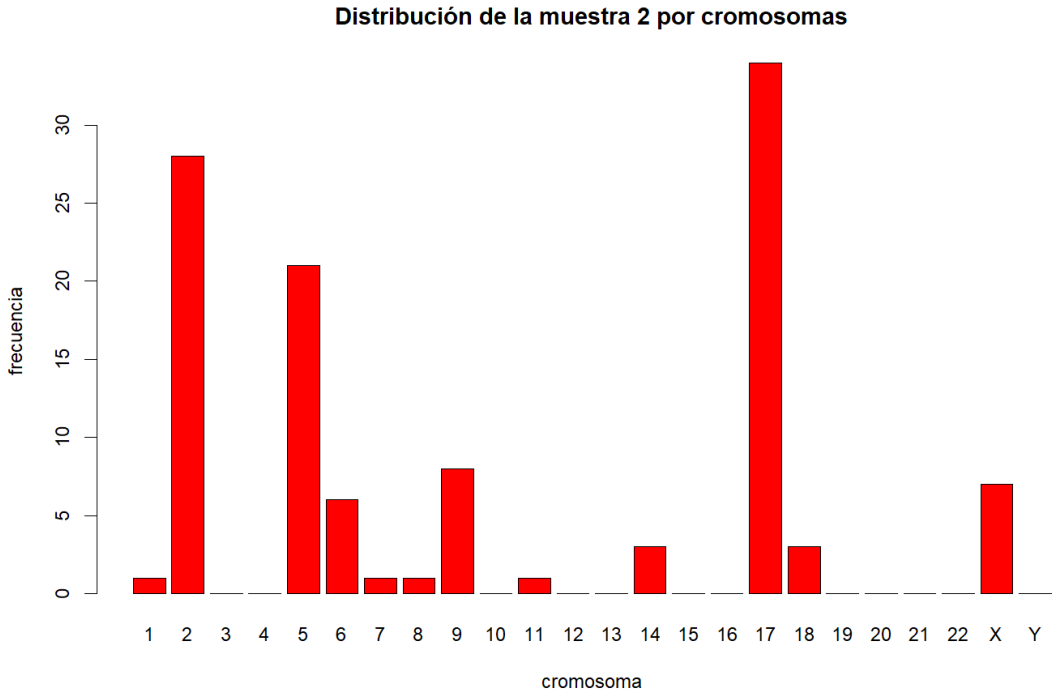

Figura 7: distribución de la muestra 2 por cromosomas

En la Tabla 2 se muestran los resultados al aplicar la función y se comparan con los obtenidos anteriormente para un conjunto de regiones generado aleatoriamente. En las Figuras 8 y 9 se muestran los histogramas de los test de permutación de ambas muestras.

Tabla 2: comparativa de resultados de clusterización de conjunto aleatorio y de conjuntos de mutaciones del cáncer de mama

Muestra	p-valor	Evaluación del conjunto
Conjuntos aleatorios	0.37	$4.52 \cdot 10^{7}$
Cáncer mama (1)	0.001	$1.47 \cdot 10^{7}$
Cáncer mama (2)	0.001	$5.36 \cdot 10^{6}$

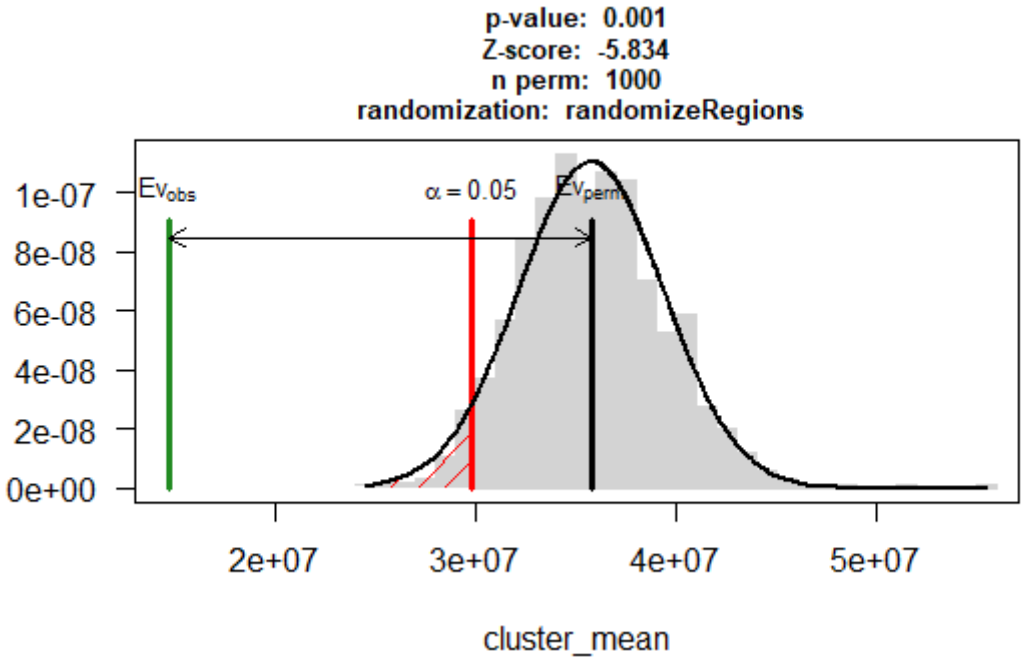

Figura 8: histograma resultante del permTest del primer conjunto de regiones de mutaciones del cáncer de mama para 3 vecinos

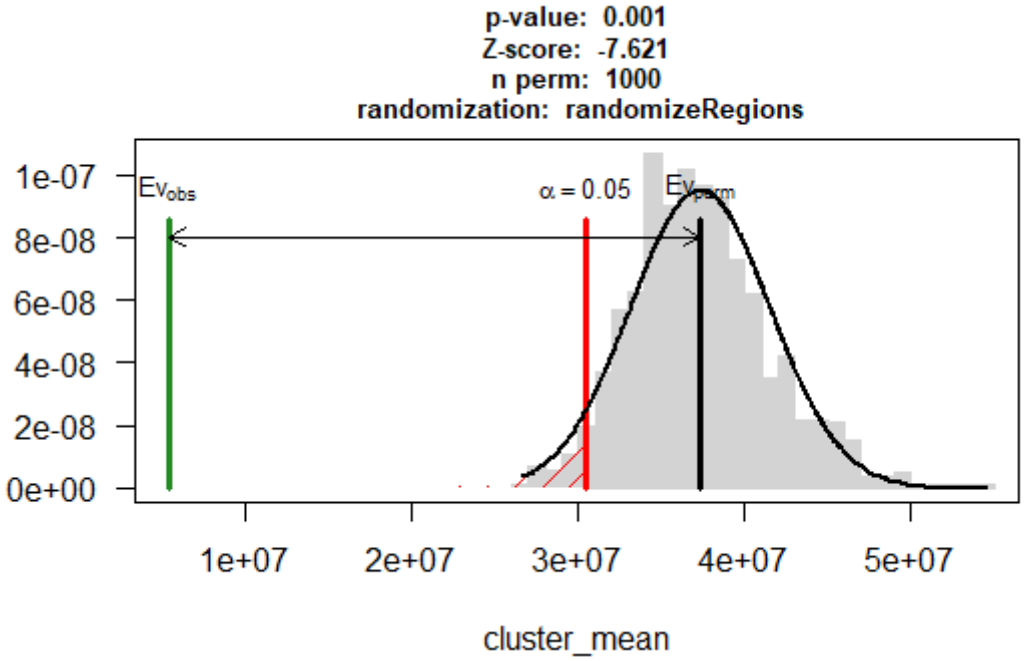

Figura 9: histograma resultante del permTest del segundo conjunto de regiones de mutaciones del cáncer de mama para 3 vecinos

Los gráficos nos muestran que, tal como se preveía, ambos conjuntos están fuertemente clusterizados. La línea roja nos indica la evaluación correspondiente a un nivel de significación de 0.05, mientras que la línea verde nos indica la evaluación de nuestra muestra.

4.3. Función *similarRegions*

El objetivo, en este caso, era obtener una función que, dado un conjunto de regiones y una función de evaluación, devolviera otro conjunto de regiones con una evaluación similar a la región dada. Este es el código de la función obtenida:

```
similarRegions<-function(A,genome="hg19",k=0.005,iter=1000,evaluate.function,...){
  # We evaluate the region for each set and get the mean
  A<-toGRanges(A)
  res<-evaluate.function(A,...)
  mean_val<-mean(res)

  # We get the length, mean and standard deviation for the given set of regions
  total_length<-length(A)
  mean_A<-mean(width(A))
  sd_A<-sd(width(A))

  # We create a random set of regions with same size, mean and standard deviation
  final_region<-
createRandomRegions(nregions=total_length,length.mean=mean_A,length.sd=sd_A)
  res1<-evaluate.function(final_region,...)
  mean_val1<-mean(res1)

  # We get the limits according to the given k to create the desired interval
  lim_inf<-mean_val-k
  lim_sup<-mean_val+k
  j<-1
  counter<-0

  # While loop will be repeateated for a given number of iterations
  # or until the mean is in the given interval
  while(counter<=iter&&(mean_val1<lim_inf||mean_val1>lim_sup)){
    # We choose the sets in the interval
    valid_index<-which (res1>=(mean_val-k)&res1<=(mean_val+k))
    valid_length<-length(valid_index)
    # We create a new set of regions with the choosen ones
    # and we complete it with random regions
    if(valid_length!=0){
      final_region[j:(j+valid_length-1)]<-final_region[valid_index]
    }
    final_region[(j+valid_length):total_length]<-
createRandomRegions(nregions=total_length-
valid_length,length.mean=mean_A,length.sd=sd_A)
    # We evaluate the new set
    res1<-evaluate.function(final_region,...)
    mean_val1<-mean(res1)
    counter<-counter+1
  }

  return(final_region)

}
```

Como se comentó anteriormente, se desarrollaron dos funciones de evaluación a modo de ejemplo. La primera de ellas, dada una región, devuelve el porcentaje de purinas que hay en ella. A continuación, se presenta el código:

```
purines_percentage<-function(A,genome=BSgenome.Hsapiens.UCSC.hg19,...){
  y<-getSeq(genome,A)
  regions<-vector(length=length(A))
  results<-vector(length=length(A))
  for (i in 1:length(regions)){
    regions[i]<-y[i]
    results[i]<-
(str_count(regions[i],"A")+str_count(regions[i],"G"))/str_length(regions[i])
  }
  return (results)
}
```

La segunda función, dado un conjunto de regiones y una secuencia, devuelve el porcentaje de bases en los que se encuentra dicha secuencia. A continuación, se presenta el código:

```
sequence_percentage<-function(A,sequence,genome=BSgenome.Hsapiens.UCSC.hg19,...){
  y<-getSeq(genome,A)
  regions<-vector(length=length(A))
  results<-vector(length=length(A))
  for (i in 1:length(regions)){
    regions[i]<-y[i]
    results[i]<-
(str_count(regions[i],sequence)*str_length(sequence))/str_length(regions[i])
  }
  return (results)
}
```

4.4 Ejemplos de aplicación de *similarRegions*

El primer testeo de la función *similarRegions* fue para obtener un conjunto de regiones con un porcentaje de purinas similar a un conjunto inicial creado aleatoriamente, utilizando como input este conjunto creado aleatoriamente, la función de evaluación *purines_percentages* y distintos valores de tolerancia *k*.

En la Tabla 3 se presentan los resultados obtenidos, con la evaluación (la proporción de purinas) y el número de iteraciones que se tardó en obtener el resultado para el conjunto original y los conjuntos obtenidos utilizando distintos valores de tolerancia.

Tabla 3: evaluación purines_percentage para distintos valores de k

Conjunto	Evaluación	Número de iteraciones
Original	0.4521763	-
k=0.005	0.4547472	2
k=0.001	0.4521882	14
k=0.0005	0.4524108	21

Podemos comprobar como en un número pequeño de iteraciones y, por tanto, con un coste computacional bajo, se obtienen conjuntos con una evaluación similar a la original según la función *purines_percentage*.

A continuación, testeamos la función *similarRegions* pasando como inputs un conjunto generado aleatoriamente, la función de evaluación *sequence_percentage* y como secuencia *CAG*.

En la Tabla 4 se presentan los resultados obtenidos, con la evaluación (el porcentaje de *CAG*) y el número de iteraciones que se tardó en obtener el resultado para el conjunto original y los conjuntos obtenidos utilizando distintos valores de tolerancia.

Tabla 4: evaluación sequence_percentage para distintos valores de k

Conjunto	Evaluación	Número de iteraciones
Original	0.05692435	-
k=0.005	0.05657919	7
k=0.001	0.05686409	41
k=0.0005	0.05694493	217

Vemos como con conjuntos aleatorios, en pocas iteraciones se consiguen conjuntos de regiones con valores muy próximos a la región original.

Finalmente, probamos la función para conjuntos de promotores. A la función *similarRegions*, se le pasó la función de evaluación *sequence_percentage*, como secuencia "GC" y como *k* se probaron distintos valores.

En la Tabla 5 se muestran las medias de la evaluación de la función *sequence_percentage* para el conjunto aleatorio del ejemplo anterior, para el conjunto de los 100 primeros promotores descargados y para la región creada mediante *similarRegions*:

Tabla 5: evaluación sequence_percentage para secuencias GC en distintos conjuntos

Conjunto	Evaluación	Número de iteraciones
Aleatorio	0.08068221	-
Promotores	0.175523	-
SimilarRegions (k=0.05)	0.125543	14
SimilarRegions (k=0.01)	0.1656362	312
SimilarRegions (k=0.005)	0.1708461	541

En este caso, al ser regiones no aleatorias, donde el porcentaje de GC es mayor que el habitual, la función requiere más iteraciones para lograr el valor deseado, pero el coste computacional es asumible. En cualquier caso, el usuario puede decidir si prefiere una *k* menos restrictiva, con lo que se puede lograr el conjunto deseado en pocas iteraciones.

5. Conclusiones y trabajos futuros

5.1. Conclusiones

El principal objetivo de este trabajo era desarrollar, testear e integrar dos nuevas funciones en el paquete regioneR. Estas funciones respondían a una necesidad, dado que dotaban al paquete de unas funcionalidades de análisis de datos biológicos que antes no tenía y que habían sido solicitadas por algunos usuarios. Por un lado, se ha diseñado una primera función, *clusteringScore*, que, dado un conjunto de regiones devuelve un índice indicando cómo de agrupadas están dentro del genoma. Por otro lado, se ha aportado una segunda función, *similarRegions*, que, dado un conjunto de regiones y una función de evaluación, devuelve otro conjunto de regiones similares según dicha función de evaluación. En este sentido, se puede decir que ambos objetivos se han logrado, ya que estas funciones han sido completamente desarrolladas, testeadas y validadas, y se han integrado en el paquete regioneR.

En relación con el testeo y ejemplos de aplicación, la función *clusteringScore* fue testeada con éxito en conjuntos de regiones aleatorias, probando distintos números de vecinos. Como era de esperar, cuantos más vecinos se tenían en cuenta, mayor era el valor de evaluación de la función, como se puede observar en la Tabla 1. Posteriormente, se testeó con un conjunto de regiones biológicas reales fuertemente clusterizadas y se obtuvo un valor mucho más bajo que el esperado para regiones aleatorias, como se puede observar en las Figuras 8 y 9. De hecho, se obtuvo en ambos casos el p-valor mínimo posible para un test con 1000 iteraciones, es decir, el test de permutación elaboró 1000 conjuntos aleatorios y ninguno obtuvo un valor de clusterización tan bajo como el del conjunto dado. Este dato era esperable dado que las regiones de estos conjuntos se encontraban concentradas en unos pocos genes, como se observa en las Figuras 6 y 7. El hecho de que el p-valor obtenido sea el mínimo posible con 1000 iteraciones muestra que este índice es una limitación para distinguir entre regiones muy clusterizadas. Una posible solución, para conjuntos con un número similar de regiones, es observar el valor de evaluación. Así, según se observa en la Tabla 2, podríamos concluir que, según el valor de evaluación, el segundo conjunto de regiones sobre las mutaciones del cáncer de mama está más clusterizado que el primer conjunto a pesar de que el p-valor es el mismo.

Respecto a la función *similarRegions*, primeramente se testeó con conjuntos de regiones aleatorias y con la función de evaluación *purines_percentage*. En este caso se obtenían buenos resultados incluso para *k* muy pequeñas, como se puede ver en la Tabla 3. Este resultado es lógico dado que se está evaluando el porcentaje de purinas en regiones aleatorias del genoma.

No hay ningún motivo para pensar que, si se seleccionan otras regiones aleatorias del genoma, el porcentaje de purinas deba variar ostensiblemente. Un caso similar sucede cuando evaluamos mediante la función *sequence_percentage* y una secuencia aleatoria, si bien, para *k* muy pequeñas ya necesita un mayor número de iteraciones, como se puede observar en la Tabla 4.

Más interesante es la interpretación de los resultados en el caso de los promotores. En la Tabla 5, se puede observar cómo el porcentaje de secuencias GC en un conjunto de promotores es el doble que en un conjunto de regiones aleatorias. Es por eso por lo que, al ir haciendo iteraciones y seleccionando las regiones con mayor porcentaje de secuencias GC, poco a poco el valor de la evaluación se va acercando a la evaluación de la región de promotores. Aquí se puede apreciar uno de los puntos fuertes de la función *similarRegions*: el usuario, mediante el factor de tolerancia k, puede decidir el equilibrio entre la similitud del conjunto devuelto con el conjunto original y el coste computacional. Otro de los puntos fuertes de *similarRegions* es la flexibilidad que ofrece al poderse utilizar con cualquier función de evaluación que devuelva un valor numérico. Para este trabajo se han diseñado dos funciones a modo de ejemplo, pero serviría igualmente con cualquier otra función de evaluación que se pasase como parámetro.

Finalmente, hay que destacar que las funciones de este trabajo han sido desarrolladas con R, un lenguaje gratuito y de código abierto. Por ello, cualquier usuario, además de utilizar las funciones, puede acceder al código, proponer posibles mejoras o personalizarlo para su uso particular.

La principal limitación de este trabajo es que ha faltado tiempo para poder probar las funciones con más conjuntos biológicos y así poder estudiar las diferentes situaciones que pudieran ir surgiendo. Por tanto, los trabajos futuros deberían ir encaminados en este sentido para poder sacar el máximo provecho de estas funciones y aplicarlas a situaciones reales.

En este trabajo se han desarrollado funciones para cuantificar el grado de clusterización de conjuntos de regiones genómicas y para obtener conjuntos aleatorios que cumplan ciertas características de acuerdo con una función de evaluación. Estas funciones son de código abierto, generalizables y personalizables para adaptarse a las necesidades de cualquier usuario, facilitando así los análisis de datos biológicos. Por tanto, este trabajo ha aportado nuevas funciones que complementan el paquete regioneR, contribuyendo así a mejorar el análisis estadístico de conjuntos de regiones genómicas.

5.2. Trabajos Futuros

Como se comentó anteriormente, los trabajos futuros deberían ir encaminados a la aplicación de las funciones desarrolladas para analizar conjuntos de regiones biológicas reales. Por un lado, para la función *clusteringScore*, se podría analizar cómo influyen el distinto número de vecinos utilizados. En este trabajo ya se hizo una aproximación para regiones generadas aleatoriamente, pero se debería estudiar con datos reales y ver si habría variación a la hora de obtener el p-valor o si se podría considerar que hay un numero de vecinos recomendado para cada caso.

La función *clusteringScore* también abre la posibilidad a trabajos relacionados con temas biológicos. Esta función podría dar respuesta a preguntas como ¿cómo están de agrupados los genes que afectan a una determinada enfermedad? o ¿los genes que contribuyen a una determinada función biológica tienden a estar más agrupados? La función *clusteringScore* puede dar respuestas numéricas que ayuden a objetivar este tipo de cuestiones.

Por otro lado, la función *similarRegions* resuelve la cuestión planteada, pero lo hace de una manera un tanto primaria. Simplemente crea regiones aleatorias, elige las que le conviene, descarta las otras y vuelve a crear regiones nuevas y repetir el proceso. Surge la pregunta ¿podría haber algún método para crear estas regiones de una manera más rápida y, por tanto, con un menor coste computacional? El principal problema que podría surgir ante cualquier método de selección de los conjuntos es que, si el método fuera muy dirigido dejarían de ser conjuntos aleatorios y, por tanto, los resultados podrían no ser válidos.

6. Glosario

Bioconductor: repositorio de paquetes de R, que nació como un proyecto de desarrollo de software gratuito para facilitar el análisis de datos de ensayos biológicos.

Clusterizar: agrupar.

Función de evaluación: función que recibe un conjunto de datos y devuelve un valor numérico.

Promotor: región de ADN que se encuentra justo antes de iniciar la transcripción de un gen.

R: lenguaje de programación, gratuito y de código abierto, muy utilizado como herramienta matemática de análisis de datos.

R Studio: entorno de desarrollo integrado para el lenguaje de programación R.

RegioneR: paquete de R/Bioconductor que, a partir de test de permutación, establece relaciones entre conjuntos de regiones genómicas y posibles características de estas.

Test de permutación: test estadístico que compara dos o más muestras para comprobar si lo que observamos se debe al azar.

7. Bibliografía

[1] V. S. Rao, S. K. Das, V. J. Rao, and G. Srinubabu, “Recent developments in life sciences research: Role of bioinformatics,” *Afr J Biotechnol*, vol. 7, no. 5, pp. 495–503, Aug. 2010, doi: 10.4314/ajb.v7i5.58463.

[2] R. Molidor, A. Sturn, M. Maurer, and Z. Trajanoski, “New trends in bioinformatics: from genome sequence to personalized medicine,” *Exp Gerontol*, vol. 38, no. 10, pp. 1031–1036, Oct. 2003, doi: 10.1016/S0531-5565(03)00168-2.

[3] “Bioconductor - Home.” https://www.bioconductor.org/ (accessed Jun. 05, 2023).

[4] P. Onghena, “Randomization Test or Permutation Test? A Historical and Terminological Clarification,” *Randomization, Masking, and Allocation Concealment*, pp. 209–228, Oct. 2017, doi: 10.1201/9781315305110-14.

[5] B. Gel, A. Díez-Villanueva, E. Serra, M. Buschbeck, M. A. Peinado, and R. Malinverni, “regioneR: an R/Bioconductor package for the association analysis of genomic regions based on permutation test,” *Bioinformatics*, vol. 32, no. 2, pp. 289–291, Jan. 2016, doi: 10.1093/BIOINFORMATICS/BTV562.

[6] A. Camargo, F. Azuaje, H. Wang, and H. Zheng, “Permutation - Based statistical test for multiple hypotheses,” *Source Code Biol Med*, vol. 3, no. 1, pp. 1–8, Oct. 2008, doi: 10.1186/1751-0473-3-15/TABLES/2.

[7] X. Gao, L. C. Becker, D. M. Becker, J. D. Starmer, and M. A. Province, “Avoiding the high Bonferroni penalty in genome-wide association studies,” *Genet Epidemiol*, vol. 34, no. 1, pp. 100–105, Jan. 2010, doi: 10.1002/GEPI.20430.

[8] M. Belmonte and D. Yurgelun-Todd, “Permutation testing made practical for functional magnetic resonance image analysis,” *IEEE Trans Med Imaging*, vol. 20, no. 3, pp. 243–248, Mar. 2001, doi: 10.1109/42.918475.

[9] S. V. Faraone, J. Su, L. Taylor, M. Wilcox, P. Van Eerdewegh, and M. T. Tsuang, “A Novel Permutation Testing Method Implicates Sixteen Nicotinic Acetylcholine Receptor Genes as Risk Factors for Smoking in Schizophrenia Families,” *Hum Hered*, vol. 57, no. 2, pp. 59–68, Jun. 2004, doi: 10.1159/000077543.

[10] “Bioconductor - BiocViews.” https://bioconductor.org/packages/release/BiocViews.html#___Software (accessed Jun. 09, 2023).

[11] “permTest: Permutation Test in regioneR: Association analysis of genomic regions based on permutation test.” https://rdrr.io/bioc/regioneR/man/permTest.html (accessed Jun. 10, 2023).

[12] J. J. Bruscadin *et al.*, “Differential Allele-Specific Expression Revealed Functional Variants and Candidate Genes Related to Meat Quality Traits in B. indicus Muscle,” *Genes (Basel)*, vol. 13, no. 12, p. 2336, Dec. 2022, doi: 10.3390/GENES13122336/S1.

[13] S. Chen *et al.*, “Comparative epigenomics reveals the impact of ruminant-specific regulatory elements on complex traits,” *BMC Biol*, vol. 20, no. 1, pp. 1–20, Dec. 2022, doi: 10.1186/S12915-022-01459-0/FIGURES/7.

[14] E. Meduri, C. Breeze, L. Marando, S. E. Richardson, and B. J. P. Huntly, “The RNA editing landscape in acute myeloid leukemia reveals associations with disease mutations and clinical outcome,” *iScience*, vol. 25, no. 12, Dec. 2022, doi: 10.1016/j.isci.2022.105622.

[15] “createRandomRegions: Create Random Regions in regioneR: Association analysis of genomic regions based on permutation test.” https://rdrr.io/bioc/regioneR/man/createRandomRegions.html (accessed Jun. 10, 2023).

[16] C. Marchal *et al.*, “High-resolution genome topology of human retina uncovers super enhancer-promoter interactions at tissue-specific and multifactorial disease loci,” *Nat Commun*, vol. 13, no. 1, Dec. 2022, doi: 10.1038/S41467-022-33427-1.

[17] “RStudio IDE - RStudio.” https://www.rstudio.com/tags/rstudio-ide/ (accessed Jun. 10, 2023).

[18] “regioneR source: R/permTest.R.” https://rdrr.io/bioc/regioneR/src/R/permTest.R (accessed Jun. 06, 2023).

[19] “Promoter.” https://www.genome.gov/genetics-glossary/Promoter (accessed Jun. 11, 2023).

[20] P. Akan and P. Deloukas, “DNA sequence and structural properties as predictors of human and mouse promoters,” *Gene*, vol. 410, no. 1–2, p. 165, Feb. 2008, doi: 10.1016/J.GENE.2007.12.011.

[21] “Bioconductor - BiSeq.” https://bioconductor.org/packages/release/bioc/html/BiSeq.html (accessed Jun. 11, 2023).