Práctica 6:

Introducción a la búsqueda de información y la minería de datos

Andrea Ferradas Agudo
Jorge Hernández Aznar
Mario Clavero Artal
Sistemas de Información 2024/2025

Índice

Desarrollo de la práctica

En esta práctica se han realizado búsquedas en documentos mediante Lucene, que es el apartado de recuperación de la información; y minería de datos mediante Weka, lo que corresponde al análisis de información de un conjunto de datos estructurados.

Para el desarrollo del programa de indexación y búsqueda de documentos de un directorio se ha utilizado el programa proporcionado para las pruebas de Lucene y se han añadido funciones auxiliares. Entre ellas destaca *buscadorFichs(fichs, path)* mediante la cual se guarda en *fichs* todos los ficheros a indexar dentro del directorio de ruta *path*, y *menuInicial()* para mostrar las posibles opciones a seleccionar por el usuario.

Después, en el *main* se ha realizado un bucle para realizar las correspondientes acciones según la elección del usuario, el cual continúa hasta que el usuario seleccione la opción 3, que es la correspondiente a terminar el programa.

Respuestas y reflexiones del bloque 1 (Lucene)

Al ejecutar sin realizar modificaciones en el fichero se han obtenido los siguientes resultados:

```
Buscando contaminación: Encontrados 4 hits.
1. ./ficheros/uno.txt      1.8415825
2. ./ficheros/uno.txt      1.8415825
3. ./ficheros/uno.txt      1.8415825
4. ./ficheros/uno.txt      1.8415825
Buscando cambio climático: Encontrados 4 hits.
1. ./ficheros/cuatro.txt   1.1087191
2. ./ficheros/cuatro.txt   1.1087191
3. ./ficheros/cuatro.txt   1.1087191
4. ./ficheros/cuatro.txt   1.1087191
Buscando por: Encontrados 4 hits.
1. ./ficheros/uno.txt      0.033158693
2. ./ficheros/uno.txt      0.033158693
3. ./ficheros/uno.txt      0.033158693
4. ./ficheros/uno.txt      0.033158693
Buscando Aeropuerto: Encontrados 4 hits.
1. ./ficheros/uno.txt      2.6491055
2. ./ficheros/uno.txt      2.6491055
3. ./ficheros/uno.txt      2.6491055
4. ./ficheros/uno.txt      2.6491055
```

Se plantean varias variaciones:
- ¿Qué pasa si utilizamos el “StandardAnalyzer” en lugar del “SimpleAnalyzer”? ¿Qué función tiene el fichero “stopwords.txt”?

El fichero stopwords.txt recoge distintas preposiciones y conjunciones usadas frecuentemente en los ficheros pero que no tienen significado semántico y por lo tanto no tienen importancia en la búsqueda. Por ello la búsqueda de la palabra “por” no obtiene resultados ya que no se tiene en cuenta.

```
Buscando contaminación: Encontrados 4 hits.
1. ./ficheros/uno.txt      2.030962
2. ./ficheros/uno.txt      1.8286357
3. ./ficheros/uno.txt      1.8286357
4. ./ficheros/uno.txt      1.8286357

Buscando cambio climático: Encontrados 4 hits.
1. ./ficheros/cuatro.txt   1.1316898
2. ./ficheros/cuatro.txt   1.098638
3. ./ficheros/cuatro.txt   1.098638
4. ./ficheros/cuatro.txt   1.098638

Buscando por: Encontrados 0 hits.

Buscando Aeropuerto: Encontrados 4 hits.
1. ./ficheros/uno.txt      2.7280912
2. ./ficheros/uno.txt      2.6493413
3. ./ficheros/uno.txt      2.6493413
4. ./ficheros/uno.txt      2.6493413
```

- ¿Qué ocurre si en la búsqueda ponemos “contaminacion” o “cambio climatico” (sin tildes)?

Al poner “contaminación” sin tilde no logra encontrar ningún fichero que tenga coincidencia ya que evalúa el carácter ‘ó’ como uno completamente distinto a ‘o’. No obstante, “cambio climático” sin tilde si que es capaz de localizarlo. Creemos que esto se puede deber a que evalúa cambio climático como una palabra compuesta y es capaz de identificar el patrón a pesar de no coincidir en la ‘a’.

```
Buscando contaminacion: Encontrados 0 hits.

Buscando cambio climatico: Encontrados 4 hits.
1. ./ficheros/cuatro.txt   0.5621123
2. ./ficheros/cuatro.txt   0.5621123
3. ./ficheros/cuatro.txt   0.5452969
4. ./ficheros/cuatro.txt   0.5452969

Buscando por: Encontrados 0 hits.

Buscando Aeropuerto: Encontrados 4 hits.
1. ./ficheros/uno.txt      2.730032
2. ./ficheros/uno.txt      2.730032
3. ./ficheros/uno.txt      2.6492126
4. ./ficheros/uno.txt      2.6492126
```

- ¿Y si hacemos esta búsqueda utilizando el “SpanishAnalyzer”? ¿Por qué ocurre esto?

En el caso del SpanishAnalyzer se incluye una conversión de todas las vocales con tilde a sus vocales sin tilde equivalentes. Esta transformación la realiza el ASCIIFoldingFilter. De esta manera puede encontrar las búsquedas pese a no contener la acentuación.

```
Buscando contaminacion: Encontrados 1 hits.
1. ./ficheros/uno.txt      5.1310577

Buscando cambio climatico: Encontrados 3 hits.
1. ./ficheros/cuatro.txt   9.638874
2. ./ficheros/dos.txt      8.134866
3. ./ficheros/tres.txt     7.9122443

Buscando por: Encontrados 0 hits.

Buscando Aeropuerto: Encontrados 1 hits.
1. ./ficheros/uno.txt      6.8034225
```

- ¿Qué ocurre si re-indexamos todos los ficheros cada vez que ejecutamos el programa, en lugar de, simplemente, reabrir el índice creado previamente?

Al re-indexar todos los ficheros cada vez que se lanza el buscador, aumentaría el tiempo de ejecución ya que tendría que leer todos los archivos, para crear los índices con los tokens necesarios. Además si no se eliminan correctamente los índices previos ya existentes, puede estar duplicando todo el contenido.

Indexación y búsqueda de documentos de un directorio

Sobre el código proporcionado en *IndexadorYBuscador* se ha desarrollado *IndexadorYBuscadorP6* en el que se ofrece un menú inicial con varias opciones a elegir por el usuario:

```
Seleccione una opción:
1.-Indexar un directorio
2.-Buscar término
3.-Salir
```

Algunas de las pruebas realizadas para comprobar el correcto funcionamiento han sido:

- 1: Indexar un nuevo directorio

```
1
Introducir directorio:
nuevo
Índice creado correctamente
```

- 2: Indexar un directorio existente

```
1
Introducir directorio:
covid
Ya existe el directorio introducido
```

- 3: Realizar distintas búsquedas (se muestran 6 ficheros como muestra de la solución)

    - COVID

```
2
Introducir directorio:
covid
Introducir término:
COVID

Buscando COVID: Encontrados 127 hits.
4. ./covid/1.txt    0.20037068
23. ./covid/4.txt   0.19741103
55. ./covid/102.txt 0.19178703
78. ./covid/117.txt 0.18595055
99. ./covid/53.txt  0.1742533
108. ./covid/16.txt 0.16856426
```

    - coronavirus

```
2
Introducir directorio:
covid
Introducir término:
coronavirus

Buscando coronavirus: Encontrados 119 hits.
1. ./covid/43.txt   0.33935422
19. ./covid/11.txt  0.33097923
47. ./covid/40.txt  0.31421077
66. ./covid/37.txt  0.29738936
92. ./covid/65.txt  0.25446263
110. ./covid/94.txt 0.22153701
```

- Salud Pública

```
2
Introducir directorio:
covid
Introducir término:
Salud Pública

Buscando Salud Pública: Encontrados 111 hits.
13. ./covid/10.txt  1.5298508
35. ./covid/134.txt 1.374243
51. ./covid/80.txt  1.2529087
67. ./covid/108.txt 1.11445
89. ./covid/33.txt  0.6143773
103. ./covid/65.txt 0.48549014
```

- pandemia

```
2
Introducir directorio:
covid
Introducir término:
pandemia

Buscando pandemia: Encontrados 97 hits.
8. ./covid/127.txt  0.736863
17. ./covid/89.txt  0.72091234
36. ./covid/77.txt  0.6834043
61. ./covid/39.txt  0.59229326
77. ./covid/91.txt  0.53606933
93. ./covid/79.txt  0.3619834
```

- 4: Salir de la ejecución

```
3
Se ha salido correctamente
```

El analizador utilizado para desarrollar el programa ha sido el SpanishAnalyzer ya que era el más adecuado para los ficheros proporcionados debido a las tildes.
Para las pruebas se han empleado palabras con bastante relación el tema del COVID-19 para así ver una gran cantidad de hits, ya que la cifra de hits es el número de documentos en los que aparece el término buscado.

Explicación desarrollo bloque 2 (Weka)

Clasificador de flores IRIS

Para la primera parte vamos a realizar un clasificador de flores en base a la información de un fichero con datos sobre tres tipos de flores IRIS, setosa, versicolor y virginica. Este fichero almacena información sobre otros 4 parámetros, a partir de esto queremos inferir de qué tipo es una flor dada según sus parámetros.

Si entrenamos el clasificador con el algoritmo *ZeroR* optará por clasificar según los datos más repetidos, en este caso como hay un 33% de cada clase predice todos los datos como pertenecientes a la primera clase, lo que supone un acierto de un tercio.

```
=== Confusion Matrix ===
  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
 50  0  0 |  b = Iris-versicolor
 50  0  0 |  c = Iris-virginica
```

Sin embargo, si cambiamos el algoritmo y lo entrenamos con *J48* los resultados obtenidos son mucho mejores. El clasificador tiene una precisión del 96% habiendo acertado 144 especies de las 150, como se puede observar en la diagonal de la matriz de confusión.

```
=== Confusion Matrix ===
  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

Cabe mencionar que para los anteriores algoritmos habían sido utilizados un 90% de los datos para entrenamiento y un 10% para la validación. Si por ejemplo disminuimos los datos de entrenamiento hasta un 50% podemos observar peores resultados ya que no tiene tantos ejemplos en los que basarse a la hora de clasificar. En este caso con un número de folds de 2 habría un 93% de precisión.

```
=== Confusion Matrix ===
  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 45  5 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

Reglas de asociación

Para la segunda parte vamos a analizar los datos de un fichero que ofrece información sobre los días que ha sido posible jugar a tenis en función de la meteorología. Concretamente disponemos de las variables *pronóstico,* siendo soleado, nublado y lluvioso, *temperatura, humedad* y *ventoso.* Dadas estas condiciones hay 15 días de los cuales 9 ha sido posible jugar a tenis y 6 no.

Para poder decidir qué días se podrá jugar o no a tenis vamos a intentar obtener reglas de asociación que muestren si hay alguna dependencia entre el estado climático y la posibilidad de jugar. Para ello se va a utilizar el algoritmo *Apriori* y las 10 mejores reglas obtenidas han sido las siguientes:

```
 1. outlook=overcast 4 ==> play=yes 4    conf:(1)
 2. temperature=cool 4 ==> humidity=normal 4    conf:(1)
 3. humidity=normal windy=FALSE 4 ==> play=yes 4    conf:(1)
 4. outlook=sunny play=no 3 ==> humidity=high 3    conf:(1)
 5. outlook=sunny humidity=high 3 ==> play=no 3    conf:(1)
 6. outlook=rainy play=yes 3 ==> windy=FALSE 3    conf:(1)
 7. outlook=rainy windy=FALSE 3 ==> play=yes 3    conf:(1)
 8. temperature=cool play=yes 3 ==> humidity=normal 3    conf:(1)
 9. outlook=sunny temperature=hot 2 ==> humidity=high 2    conf:(1)
10. temperature=hot play=no 2 ==> outlook=sunny 2    conf:(1)
```

De los resultados para nuestro caso los más determinantes son los que extraen las conclusiones sobre jugar o no. En el primero podemos observar que cuando el día está nublado se ha podido jugar así que podríamos extraer la conclusión de que el clima idóneo para jugar a tenis es cuando está nublado. Del mismo modo para la tercera regla cuando la humedad es normal y no hace viento ha habido 4 ocasiones en las que se ha podido jugar también. Se puede ver también que cuando está soleado y la humedad es alta no es recomendable jugar a tenis.

Además hay que tener en cuenta que hemos seleccionado un máximo de diez reglas posibles, también tenemos un límite inferior de soporte de 0.1, en cambio si lo aumentamos nos mostraría menos reglas ya que es más exigente para aceptar un regla. Asimismo la confianza mínima está establecida en 0.9 para mostrar una regla si la disminuimos nos muestra más reglas pese a que no tengan tanta certeza, por ejemplo poniendo minMetric a 0.5 una de las reglas es la siguiente

```
 4. humidity=normal 7 ==> play=yes 6    conf:(0.86)
```

Es muy similar a la tercera anterior, en este caso como de los 7 días que ha habido una humedad normal ha podido jugar 6 de ellos es muy probable que cuando haga una humedad media pueda ser posible jugar a tenis.

Análisis de sentimientos y lenguaje natural

En la tercera parte vamos a analizar un conjunto de opiniones sobre películas y vamos a intentar determinar a partir del lenguaje natural si son reviews positivas o negativas. Tras ejecutar el algoritmo de *naive bayes* los resultados son los siguientes:

```
=== Detailed Accuracy By Class ===
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.75      0.185      0.802     0.75      0.775      0.869    pos
                 0.815     0.25       0.765     0.815     0.789      0.869    neg
Weighted Avg.    0.783     0.218      0.784     0.783     0.782      0.869

=== Confusion Matrix ===
   a   b   <-- classified as
 750 250 |   a = pos
 185 815 |   b = neg
```

Inicialmente las opiniones detectadas como positivas correctamente son 750, las detectadas como positivas incorrectamente son 185, los aciertos negativos son 815 y las clasificadas como negativas que son realmente positivas son 250.

La ***precisión*** para detectar una clase positiva ha sido del 80.2% (= 750 /(750+185)) ya que se calcula en función de los resultados predichos como positivos que son realmente positivos. No obstante la precisión para las opiniones negativas son ligeramente menores siendo de 76.5% (= 815/(815+250)) ya que tiene una pequeña tendencia a clasificar más opiniones como negativas. Siendo el total de opiniones predichas como positivas 935 y negativas como 1065.

El ***recall*** indica la cantidad de cantidad de opiniones positivas predecidas frente a la cantidad de opiniones positivas reales. En las reviews positivas es más baja que en las negativas ya que como acabamos de comentar el clasificador tiene tendencia a predecir más opciones como negativas. El recall positivo es del 75% (= 750/(750+250)). Por otro lado el recall negativos es más de un 6% más alto siendo de 81.5% (= 815/(815+185)).

En ambos casos tanto la precisión como el recall promedio es la media entre los positivos y negativos ya que los datos son los mismos para los positivos que para los negativos. Siendo la precisión de 78.4% y el recall de 78.3%.

Tras eliminar palabras sin significado semántico como '*', '-' o '+' se obtiene estos resultados:

```
=== Detailed Accuracy By Class ===
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.753     0.189      0.799     0.753     0.775      0.869    pos
                 0.811     0.247      0.767     0.811     0.788      0.869    neg
Weighted Avg.    0.782     0.218      0.783     0.782     0.782      0.869

=== Confusion Matrix ===
   a   b   <-- classified as
 753 247 |   a = pos
 189 811 |   b = neg
```

Podemos ver que aumentan ligeramente la tendencia a predecir cómo reviews positivas, no obstante pese a clasificar 3 verdaderas positivas más hay 4 menos clasificadas de forma correcta en las negativas. Pese a aumentar la precisión de las clasificadas como negativas y el recall de las positivas las métricas contrarias disminuyen de forma casi equivalente, incluso empeorando mínimamente la precisión y el recall promedio siendo 0.1% peor respectivamente, 78.3% y 78.2% respectivamente frente a los 78.4% y 78.3% previos.

Una forma alternativa de obtener mejores resultados podría ser aumentar el número de divisiones poniendo 20 folds en vez de 10 por ejemplo para aumentar el número de datos de entrenamiento. En ese caso los resultados serían los siguientes.

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.787     0.166      0.826     0.787     0.806      0.899    pos
                 0.834     0.213      0.797     0.834     0.815      0.899    neg
Weighted Avg.    0.811     0.19       0.811     0.811     0.81       0.899

=== Confusion Matrix ===

   a   b   <-- classified as
 787 213 |   a = pos
 166 834 |   b = neg
```

En este caso la precisión y el recall promedio mejora hasta el 81.1% para ambos.

Profundizando en los árboles de clasificación

Para el último apartado vamos a analizar que medicamentos deberíamos recomendar según las variables de edad, sexo, tensión sanguínea(BP) , colesterol, nivel de sodio en sangre(Na) y nivel de potasio en sangre(K). Para ello disponemos de 5 medicamentos.

Si hacemos la clasificación según el algoritmo de *ZeroR* tendríamos en cuenta el medicamento más usualmente recetado que sería drugY, este tendría una precisión de 20.7% y recall de 45.5%, resultados bastante mejorables.

```
=== Detailed Accuracy By Class ===
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1          0.455     1         0.625      0.45     drugY
                 0         0          0         0         0          0.391    drugC
                 0         0          0         0         0          0.447    drugX
                 0         0          0         0         0          0.437    drugA
                 0         0          0         0         0          0.391    drugB
Weighted Avg.    0.455     0.455      0.207     0.455     0.285      0.438

=== Confusion Matrix ===
  a  b  c  d  e   <-- classified as
 91  0  0  0  0 |  a = drugY
 16  0  0  0  0 |  b = drugC
 54  0  0  0  0 |  c = drugX
 23  0  0  0  0 |  d = drugA
 16  0  0  0  0 |  e = drugB
```

Este algoritmo solo tiene un 100% de recall para la drugY ya que clasifica todas como esa y además tiene una precisión de 45,5% para esa ya que muchas clasificados como drugY están mal clasificados en concreto hay 91 bien clasificados y 109 mal clasificados.

Para mejorar este algoritmo podemos usar el usado previamente el *J48,* el cual con 10 folds presenta los siguientes resultados:

```
=== Detailed Accuracy By Class ===
               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.945     0.073      0.915     0.945     0.93       0.941    drugY
                 0.938     0.016      0.833     0.938     0.882      0.959    drugC
                 0.907     0.014      0.961     0.907     0.933      0.952    drugX
                 0.913     0.011      0.913     0.913     0.913      0.951    drugA
                 0.875     0          1         0.875     0.933      0.932    drugB
Weighted Avg.    0.925     0.04       0.927     0.925     0.925      0.946

=== Confusion Matrix ===
  a  b  c  d  e   <-- classified as
 86  2  2  1  0 |  a = drugY
  1 15  0  0  0 |  b = drugC
  4  1 49  0  0 |  c = drugX
  2  0  0 21  0 |  d = drugA
  1  0  0  1 14 |  e = drugB
```

Este mejora radicalmente la precisión y el recall promedio con 92.7% y 92.5% respectivamente. Asimismo se puede observar como la matriz de confusión tiene predicciones mucho más realistas y equilibradas. No obstante el árbol de decisión resultante es muy complejo.

```
K <= 0.055221
|   K <= 0.037124: drugY (56.0)
|   K > 0.037124
|   |   Na <= 0.685143
|   |   |   BP = HIGH
|   |   |   |   Na <= 0.656371: drugA (6.0)
|   |   |   |   Na > 0.656371: drugY (2.0/1.0)
|   |   |   BP = LOW
|   |   |   |   Sex = F: drugC (3.0)
|   |   |   |   Sex = M: drugX (4.0/1.0)
|   |   |   BP = NORMAL: drugX (11.0/1.0)
|   |   Na > 0.685143: drugY (33.0/2.0)
K > 0.055221
|   BP = HIGH
|   |   Age <= 50: drugA (17.0)
|   |   Age > 50: drugB (15.0)
|   BP = LOW
|   |   Cholesterol = HIGH: drugC (14.0/1.0)
|   |   Cholesterol = NORMAL: drugX (13.0)
|   BP = NORMAL: drugX (26.0)
```

Para reducir la complejidad del árbol podemos observar que hay una dependencia entre los atributos K y Na, para ello creamos un atributo derivado que sea su cociente K/Na y eliminamos sus dos atributos. De este modo obtenemos mejores resultados y un árbol más sencillo.

Los resultados son los siguientes,

```
Na_to_K <= 0.0666: drugY (91.0)
Na_to_K > 0.0666
|   BP = HIGH
|   |   Age <= 50: drugA (23.0)
|   |   Age > 50: drugB (16.0)
|   BP = LOW
|   |   Cholesterol = HIGH: drugC (16.0)
|   |   Cholesterol = NORMAL: drugX (18.0)
|   BP = NORMAL: drugX (36.0)
```

El árbol dispone de únicamente 6 hojas frente a las 12 del anterior, además podemos ver que la precisión y el recall promedio suben hasta un 99%. Finalmente en la matriz de confusión hay únicamente dos resultados que se quedan fuera de la diagonal

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.989     0          1         0.989     0.994      0.995    drugY
                 1         0.005      0.941     1         0.97       0.997    drugC
                 1         0          1         1         1          1        drugX
                 1         0.006      0.958     1         0.979      0.997    drugA
                 0.938     0          1         0.938     0.968      0.969    drugB
Weighted Avg.    0.99      0.001      0.991     0.99      0.99       0.994

=== Confusion Matrix ===

  a  b  c  d  e   <-- classified as
 90  1  0  0  0 |  a = drugY
  0 16  0  0  0 |  b = drugC
  0  0 54  0  0 |  c = drugX
  0  0  0 23  0 |  d = drugA
  0  0  0  1 15 |  e = drugB
```

Metodología de trabajo empleada

Herramientas

Las herramientas utilizadas en el desarrollo de esta práctica han sido Eclipse IDE para la parte de Lucene y el desarrollo del programa de indexación y búsqueda de documentos en un directorio, y Weka para la parte de la minería de datos.

Distribución del trabajo y horas empleadas

Respecto a la distribución del trabajo, todos los integrantes del grupo hemos contribuido a cada apartado de la práctica; de esta forma cada uno ha trabajado en todos los aspectos a desarrollar.
Hemos trabajado en las sesiones de prácticas y fuera de clase de manera conjunta, por lo que las horas empleadas en total serían 8.

Aproximadamente se han dedicado 1 hora a la parte de las pruebas de Lucene, 3 horas al desarrollo del programa de indexación y búsqueda de documentos en un directorio y 4 horas a la parte de Weka.

Dificultades

Las principales dificultades encontradas han sido en la parte del desarrollo del programa ya que al momento de realizar la búsqueda dependiendo de cuantas veces se indexaba el directorio se obtenían distinto número de hits. Esto ocurría porque no se tenía en cuenta si ya había un índice creado para un directorio, entonces se sobrescribía el índice. Para ello antes de indexar un directorio se comprueba que no se haya indexado anteriormente.