Universidad Autónoma Metropolitana Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

Proyecto Terminal

SISTEMA CLASIFICADOR DE ARCHIVOS DE MÚSICA USANDO EL CONCEPTO DE MEMORIA ASOCIATIVA

Trimestre: 10-O

Alumno:

Sergio Rivera Bernal
Matrícula: 204243068

Asesor:

Dr. Oscar Herrera Alcántara

México D.F., Diciembre de 2010

Contenido

Antecedentes

El cerebro humano ha sido un gran misterio desde siempre. Aún cuando se conocen los elementos que conforman su estructura, no se sabe cuál es la interacción que existe entre ellos, lo que nos permite llevar a cabo diversas actividades y desarrollar algunas habilidades. Una de las tantas habilidades que posee nuestro cerebro es la de poder recordar, a lo que llamamos memoria.
Nuestra memoria puede ser asociativa, ya que asocia diversos objetos que posteriormente nos sirven para recordar. Por ejemplo, si no recordamos el rostro de algún conocido, entonces alguien puede ayudarnos mencionando ciertas características, como pueden ser, el color del pelo, de los ojos o de la tez de esa persona, quizá algún tatuaje o cicatriz. Alguno de esos detalles o su combinación, provocan la asociación automática con la persona que posee esas características.

Otras veces, con tan sólo escuchar una canción, percibir un olor o simplemente al escuchar el nombre de un lugar, podemos recordarlo. Por lo tanto, nuestra forma de recordar es mediante asociaciones, un pensamiento nos lleva a otro y éste hacia otro y así sucesivamente, hasta que llegamos a un pensamiento final, que quizá no tenga relación con el primer pensamiento.

En el proceso de memorización siempre intervienen dos fases:

Fase de asociación. Se produce la asociación de los objetos en cuestión el que queremos recordar (objetivo) y el que nos estimula al recuerdo (estimulante).

Fase de recuperación. Basta con alimentar a nuestra memoria con el objeto estimulante y el objetivo aparecerá automáticamente.

Lo que en realidad hace nuestra memoria es reconocer ciertas características mostradas por un objeto, lo que en el área de la Inteligencia Artificial se llama Reconocimiento de Patrones.
Sin embargo, nuestra memoria no sólo sirve para recordar objetos, también nos ayuda en la clasificación de los mismos. Basta con aprender las características que distinguen a ciertos objetos de naturaleza similar y cuando uno de ellos se nos presenta, automáticamente sabremos a que clase pertenece [1].

Con el propósito de simular el comportamiento de la memoria del ser humano, la ciencia de la computación ha desarrollado diversos modelos matemáticos de recuperación de información, dichos modelos matemáticos son conocidos como memorias asociativas en el área de la Inteligencia Artificial. Una memoria asociativa es un elemento cuyo propósito fundamental es recuperar patrones completos a partir de patrones de entrada que pueden ser alterados con ruido sustractivo, aditivo o mixto, puede almacenar información y poder recuperarla cuando sea necesario, es decir, una red retroalimentada, cuya salida se utiliza repetidamente como una nueva entrada hasta que el proceso converge [2].

El presente proyecto tiene como objetivo describir la metodología a través de la cual se realiza el reconocimiento de archivos de sonido almacenados en formato RIFF. El formato RIFF representa al formato WAV, en el que se almacenan muestras de una señal de audio en modulación por codificación de pulsos (Pulse Coding Modulation, PCM). La codificación Parson (PARSON-Code) nos permite describir el contorno de una melodía. El uso de la transformada wavelet discreta nos ayuda para la reducción del tamaño de los archivos de sonido, así como la generación de coeficientes de transformación con rasgos característicos con los que es posible distinguir archivos de sonido. Los rasgos característicos se representan como vectores *n*-dimensionales que se utilizan como entrada de una red neuronal usada como memoria asociativa.

Las pruebas realizadas en este proyecto muestran la posibilidad de realizar el reconocimiento de archivos de sonido empleando bloques de $n = 8$ coeficientes de baja frecuencia, representando cada uno de estos coeficientes a la potencia obtenida por banda de transformación.

Palabras clave: PCM, sonido, wavelet, red neuronal, reconocimiento de patrones, WEKA.

1. Introducción

La obtención de los archivos WAV se realiza a través del proceso de lectura (ripping) de las pistas de discos compactos (tracks). La calidad del sonido al reproducirlo, depende del número de muestras por segundo (frecuencia de muestreo o SampleRate). Comúnmente realizar una búsqueda de archivos de audio implica hacerla a través del título de la canción, autor, género, álbum, entre otras, para lo cual existen diferentes herramientas software que brindan estas opciones. También existen sistemas de consulta que utilizan el tarareo (Query by Humming)[3], que forman parte de los sistemas de Recuperación de Información Musical (RIM)[4] que hace posible identificar una canción por medio de una fragmento de la misma o con tarareos proporcionados a través de un micrófono. Este método de búsqueda no es nuevo, al igual del uso de wavelets con el mismo objetivo [5][6]. Este proyecto tiene como objetivo desarrollar un sistema de reconocimiento de audio utilizando codificación PARSON así como el uso de wavelet Haar. La diferencia de este proyecto con otros es que se utiliza el menor número de coeficientes de baja frecuencia, puesto que se conservan propiedades tiempo-frecuencia de la señal de sonido [7] y de esta manera se obtiene la potencia aplicada a cada nivel de transformación. Una vez adquiridos los rasgos característicos del archivo de sonido se requiere hacer búsquedas con empates parciales. El uso de una red neuronal artificial nos permite realizar este tipo de reconocimientos parciales. Sin embargo la cantidad de muestras por segundos es demasiado alta y resulta ineficiente el procesamiento a través de la red neuronal artificial. Para hacer eficiente el procesamiento es necesario reducir la dimensión del vector, bajando la frecuencia de muestreo a 6 KHz con lo que se reduce a 1,080,000 el número de muestras.

En base a las consideraciones anteriores, puede establecerse el siguiente sistema de búsqueda (figura 1.1).

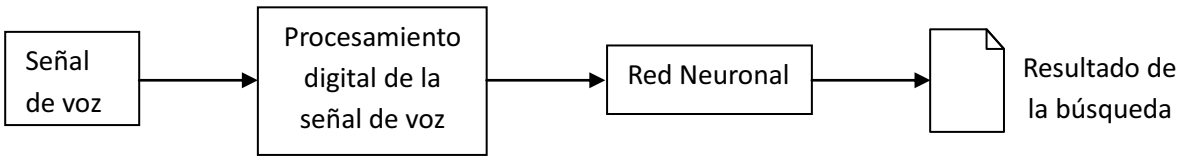

Figura 1.1: Diagrama de bloques de la arquitectura del sistema de búsqueda de canciones mediante tarareo.

La primera etapa consiste en capturar el sonido de voz (tarareo) a través de un micrófono y digitalizarla. Esta señal de audio digital se codifica PCM en formato WAV. En la segunda etapa, se realiza un procesamiento de la señal digitalizada extrayendo un conjunto de rasgos característicos. En la siguiente etapa se utiliza una arquitectura apropiada de una red neuronal que tiene como entrada el conjunto de rasgos característicos de la señal procesada en la etapa anterior. Finalmente, el sistema devuelve una lista de pistas de audio (tracks) ordenadas según su similitud con la consulta (tarareo).

1.1 Organización

A continuación se detalla la organización de este proyecto.

En el Capítulo 2, *Preprocesamiento de archivos de sonido*, se describen técnicas para la obtención de un conjunto de rasgos característicos que identifican de manera única a cada archivo de sonido con otro.
En el Capítulo 3, *Red Neuronal*, se implementará una red neuronal para la fase de entrenamiento con los datos obtenidos en la etapa anterior. Para implementar la red neuronal artificial se ha utilizado la herramienta Weka. Weka es una extensa colección de algoritmos de Máquinas de conocimiento desarrollados por la universidad de Waikato (Nueva Zelanda) implementados en Java. La licencia de Weka es GPL (GNU Public License), lo que significa que este programa es de libre distribución y difusión. De esta manera, obtendremos una arquitectura de una red neuronal adecuada para lograr una buena capacidad de generalización.
En el Capítulo 4, *Evaluación*, se presentan los experimentos realizados para la evaluación del sistema y sus resultados.
En el Capítulo 5, *Creación de aplicación Software*, en la que se presentan los resultados de las consultas y en la que se podrá seleccionar un archivo de audio para su reproducción o bien realizar más consultas.
En el Capítulo 6, *Conclusiones y trabajos futuros*, se realiza un análisis crítico al proyecto realizado y se sugieren futuros métodos de optimización para mejores resultados.

2. Preprocesamiento de archivos de audio

2.1 Codificación Parson

En algunos sistemas de búsqueda por tarareo es de suma importancia una representación de las melodías (simulando notación musical) con el objetivo de poder realizar un análisis comparativo. A esta representación de las melodías se le conoce como contorno. Una manera simple de obtener este contorno es mediante el uso de tres valores que describen los intervalos de nota a nota, estos valores son: up (U), down (D) y repeat (R)[7].

En este proyecto se utiliza codificación Parson para obtener el contorno de cada archivo de audio, modificando los valores que describen los intervalos de nota a nota ya que a diferencia de los sistemas que emplean este método no realizaremos un análisis basado en notas, los valores usados son los siguientes: up (1.0), down (0.0) y repeat (0.5).

2.2 Transformada wavelet

La transformada wavelet es una herramienta matemática relativamente reciente que surge, como técnica de ingeniería, para el análisis de series temporales, Básicamente, consiste en la realización de un análisis de la señal a diferentes niveles de resolución. A diferencia de lo que ocurre con la transformada de Fourier, la transformada wavelet no realiza una descomposición de la señal en el dominio de la frecuencia, sino que analiza la señal a diferentes niveles de resolución o escalas. La principal diferencia entre ambas transformadas es el carácter local de las wavelets, lo que probablemente constituya igualmente la mayor ventaja para el análisis de señales no estacionarias, frente al análisis global realizado por la transformada de Fourier.

La wavelet más simple de estudiar es la wavelet de Haar cuya expresión matemática es:

$$\psi(t) = \begin{cases} 1 & 0 \le t < 1/2, \\ -1 & 1/2 \le t < 1, \\ 0 & \textit{caso contrario} \end{cases}$$

Y su función de escalamiento puede ser descrita como:

$$\phi(t) = \begin{cases} 1 & 0 < t < 1. \\ 0 & \textit{caso contrario} \end{cases}$$

La transformada Haar, realiza una descomposición de la señal en dos componentes. Sean s_0 y s_1 dos muestras de un vector a transformar. La componente de baja frecuencia está dada por:

$$a = \frac{s_0 + s_1}{2}$$

en tanto que la componente de alta frecuencia está dada por:

$$c = \frac{s_0 - s_1}{2}$$

Tomando en cuenta lo mencionado podemos aplicar nuevamente la transformada wavelet sobre la mitad de los coeficientes que concentran la mayor parte de la energía (componente de baja frecuencia), y así sucesivamente hasta tener un mínimo de dos muestras. Entonces a una señal de *m* muestras se le puede aplicar un máximo de $log_2 m$ niveles de transformación.

2.3 Extracción de rasgos característicos de archivos de sonido

El formato de archivo WAV canónico [8], es un subconjunto de las especificaciones RIFF para el almacenamiento de archivos multimedia. Un archivo RIFF comienza con un encabezado de archivo seguido por una secuencia de fragmentos de datos mencionados en la tabla I.

Byte inicial	Identificador	Longitud (Bytes)
0	ChunkID	4
4	ChunkSize	4
8	Format	4
12	Subchunk1ID	4
16	Subchunk1Size	4
20	AudioFormat	2
22	NumChannels	2
24	SampleRate	4
28	ByteRate	4
32	BlockAlign	2
34	BitsPerSample	2
36	Subchunk2ID	4
40	Subchunk2Size	4
44	Datos	*

Tabla I. Encabezado de un archivo WAV canónico.

A continuación se describen los segmentos del encabezado de la Tabla I.

- ChunkID. Contiene los caracteres “RIFF” en formato ASCII.
- ChunkSize. Almacena el valor del tamaño del archivo WAV menos 8 bytes.
- Format. Contiene los caracteres “WAVE”.
- SubChunk1ID. Contiene los caracteres “fmt”.
- SubChunk1Size. Almacena el valor del bloque de datos que comienza aquí y termina antes del bloque de datos.
- AudioFormat. Un valor de 1 indica que no hay compresión de datos
- NumChannels. Mono = 1, Stereo = 2.
- SampleRate. Almacena el valor de la frecuencia de muestreo.
- ByteRate. Almacena el valor de *SampleRate * NumChannels * BitsPerSample/8.*
- BlockAlign. Almacena el valor de *NumChannels * BitsPerSample/8.*
- BitsPerSample. Indica el número de bits por muestra.
- Subchunk2ID. Contiene los caracteres “data”.
- Subchunk2Size. Almacena el valor de *NumSamples * NumChannels * BitsPerSample/8.*
- Datos. Almacena las muestras de sonido.

3. Red neuronal

Una red neuronal artificial (RNA por sus siglas), está constituida por elementos que se comportan de forma similar a una neurona biológica (neurona artificial). Estos elementos están organizados de una forma parecida a la que presenta el cerebro humano. Las tres partes fundamentales de una neurona son su entrada, su salida y su capacidad de procesamiento. Por lo tanto una neurona artificial es un modelo matemático que simula el comportamiento de una neurona biológica.

Uno de los modelos más conocidos es el que propuso Rosenblat el cual se muestra a continuación (Figura 3.1)

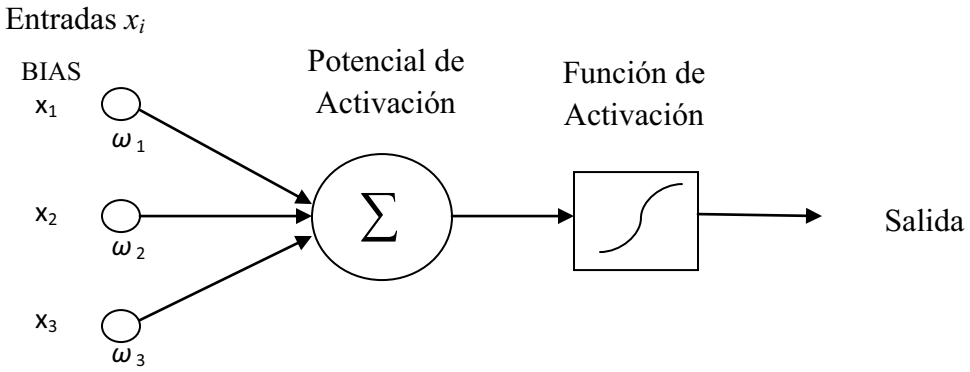

Figura 3.1. Modelo de Perceptrón propuesto por Rosenblat

A la conexión de varias salidas de perceptrones usadas como entradas de otros se le conoce como red de perceptrones. Por lo que la salida de esta red neuronal está dada por la evaluación en cascada de la salida de varios perceptrones, dando lugar a una capa oculta cuyas salidas vuelven a ser entrada de otra capa de perceptrones (Figura 3.2).

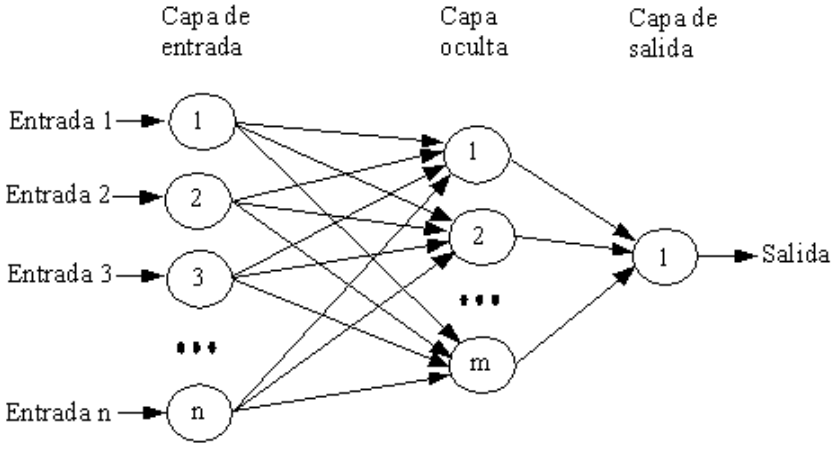

Figura 3.2. Modelo de red de Perceptrones con una capa oculta.

Para trabajar con una red neuronal existen varios procesos.

- Normalización. En este proceso se realiza un escalamiento de los datos de entrada a un intervalo [0, 1].
- Entrenamiento. Consiste en colocar nuevos valores en la entrada de la red neuronal para dar lugar a un ajuste de los pesos sinápticos y de esa manera aprender esos nuevos valores.
- Aprendizaje. Consiste en minimizar el error entre la salida deseada y la obtenida ajustando el valor de los pesos sinápticos.
- Prueba. Consiste en realizar pruebas con valores que no causan ajuste en los pesos sinápticos y así poder comprobar una salida optima
- Generalización. Consiste en colocar valores en la entrada de la red neuronal que no se usaron para el entrenamiento y obtener salidas aceptables.

4. Evaluación

Esta etapa se divide en varias fases, las cuales se describen a continuación.

Fase I. Obtención de rasgos característicos.

1. Creación de una base de datos llamada tarareo que contenga una tabla llamada potencias en la que se almacenarán los resultados obtenidos (ver sección instalación de software).
2. Iniciamos el proceso leyendo el encabezado del archivo de sonido a procesar (un total de 10 archivos WAV con duración recortada a 40 segundos) hasta llegar al bloque datos. Mostrándonos el contenido de los bloques del archivo de sonido.
3. Extraemos la información de las muestras de sonido almacenándola como números de tipo short en un arreglo realizando la conversión correspondiente puesto que los datos leídos son bytes.

4. Realizar codificación Parson a toda la información, la cual es almacenada en números de tipo double un vector.

5. Realizar cortes cada 10,9 segundos aproximadamente, generando bloques de longitud $V = 65536$ muestras (2^{17}) que están desplazadas cada $z = 200$ muestras. A cada bloque obtenido se le aplican 7 niveles de transformada wavelet Haar y en cada nivel de transformación se calcula su potencia, de esta manera se obtiene $\omega = 8$ coeficientes, los cuales están almacenados como números de tipo entero en una base de datos incluyendo al final el identificador (número del archivo / número total de archivos +1) del archivo de sonido procesado. De esta manera obtenemos en promedio $k = 873$ vectores de dimensión 9 (8 coeficientes + 1 identificador) por cada archivo de sonido procesado.

Ver código 1 en la sección códigos fuente. En el que se muestra el código completo correspondiente a esta fase.

Fase II. Recuperación de información

1. Accesar a la base de datos para obtener el máximo y el mínimo por columna de la tabla potencias de cada ω = 8 coeficientes calculados, esto con el objetivo de poder realizar el proceso de normalización.
2. Se crea la cabecera de un archivo con extensión arff (Formato de Archivo de Atributo-Relación) llamado trainWeka.arff en cual usaremos para el entrenamiento de la red neuronal.
3. Al termino de creación de la cabecera se procede a la lectura de los datos para realizar la normalización de los datos usando la fórmula (1):

$$x_i = \frac{x_i - \min}{\max - \min} \quad (1)$$

Y de esta manera poder completar la sección correspondiente a los datos en el archivo trainWeka.arff (Ver figura 4.1).

```
@relation trainWeka

@attribute nivel_1 NUMERIC
@attribute nivel_2 NUMERIC
@attribute nivel_3 NUMERIC
@attribute nivel_4 NUMERIC
@attribute nivel_5 NUMERIC
@attribute nivel_6 NUMERIC
@attribute nivel_7 NUMERIC
@attribute nivel_8 NUMERIC
@attribute id_cancion NUMERIC
@data

0.6288478452066842,0.3708154506437768,0.4492099322799097,0.5723270440251572,0.9772727272727273,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6328056288478452,0.3725321888412017,0.4492099322799097,0.5660377358490566,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6380826737027264,0.37510729613733906,0.45146726862302483,0.5723270440251572,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6429199648197009,0.3785407725321888,0.45372460496614,0.5723270440251572,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6486367634124 89,0.3811158798283262,0.4559819413092550 7,0.5786163522012578,0.9772727272727273,0.9523809523809523,0.8181818181818182,1.0,0.09
0.6521547933157432,0.38197424892703863,0.4559819413092550 7,0.5723270440251572,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6556728232189973,0.384549356223176,0.4582392776523702,0.5786163522012578,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6613896218117854,0.38798283261802574,0.4604966139954853,0.5723270440251572,0.9545454545454546,0.9047619047619048,0.8181818181818182,1.0,0.09
0.6662269129287599,0.3905579399141631,0.4627539503386004 5,0.5786163522012578,0.9772727272727273,0.9047619047619048,0.9090909090909091,1.0,0.09
0.6728232189973615,0.3948497854077253,0.4672686230248307,0.5786163522012578,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6802990325417766,0.3991416309012875 7,0.4695259593679458 3,0.5911949685534591,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6868953386103782,0.4034334763948498,0.4740406320541760 7,0.5849056603773585,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6926121372031663,0.40772532188841204,0.4762979683972912,0.5974842767295597,1.0,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6952506596306068,0.40858369098712444,0.4762979683972912,0.5911949685534591,1.0,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6952506596306068,0.41030042918454934,0.4785553047404063,0.5911949685534591,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6948109058927001,0.40858369098712444,0.4762979683972912,0.5849056603773585,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6956904133685137,0.4094420600858369,0.4762979683972912,0.5911949685534591,1.0,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6956904133685137,0.40858369098712444,0.4762979683972912,0.5786163522012578,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6978891820580475,0.4094420600858369,0.4762979683972912,0.5849056603773585,0.9545454545454546,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6996481970096746,0.4111587982832618,0.4762979683972912,0.5786163522012578,0.9772727272727273,0.9523809523809523,0.9090909090909091,1.0,0.09
0.6992084432717678,0.4094420600858369,0.4740406320541760 7,0.5786163522012578,0.9772727272727273,0.9047619047619048,0.8181818181818182,1.0,0.09
```

Figura 4.1. Formato de archivo trainWeka.arff generado.

Ver código 2 en la sección códigos fuente. En el que se muestra el código completo correspondiente a esta fase.

Fase III. Implementación de red neuronal

1. Para la implementación de la red neuronal se utilizo la API (interfaz de programación de aplicaciones) WEKA, esta API contiene un modelo de red neuronal (Multilayer Perceptron), que nos permite realizar el entrenamiento de la misma.
   Para realizar el entrenamiento de la red neuronal, se necesita el archivo generado en la fase anterior (trainWeka.arff) y definir una arquitectura adecuada para un óptimo entrenamiento. En este caso con 2500 épocas se logró minimizar el error de entrenamiento. Los datos obtenidos por WEKA se muestran en la tabla II:

Correlation coefficient	0.9948
Mean absolute error	0.0194
Root mean squared error	0.0273
Relative absolute error	8.4154 %
Root relative squared error	10.3959 %
Total Number of Instances	8730

Tabla II. Resultados generados por WEKA al finalizar el entrenamiento para 10 canciones.

En la tabla III se muestra la arquitectura de la red neuronal entrenada así como sus parámetros de entrenamiento y en la figura 4.2 se muestra la red neuronal generada por WEKA.

Capa	Neuronas	Función de Activación	Tasa de aprendizaje	Momentum
Entrada	8	Lineal	0.1	0.1
Oculta 1	30	Sigmoidal	0.1	0.1
Oculta 2	26	Sigmoidal	0.1	0.1
Salida	1	Lineal	0.1	0.1

Tabla III. Arquitectura de red neuronal entrenada para 10 canciones.

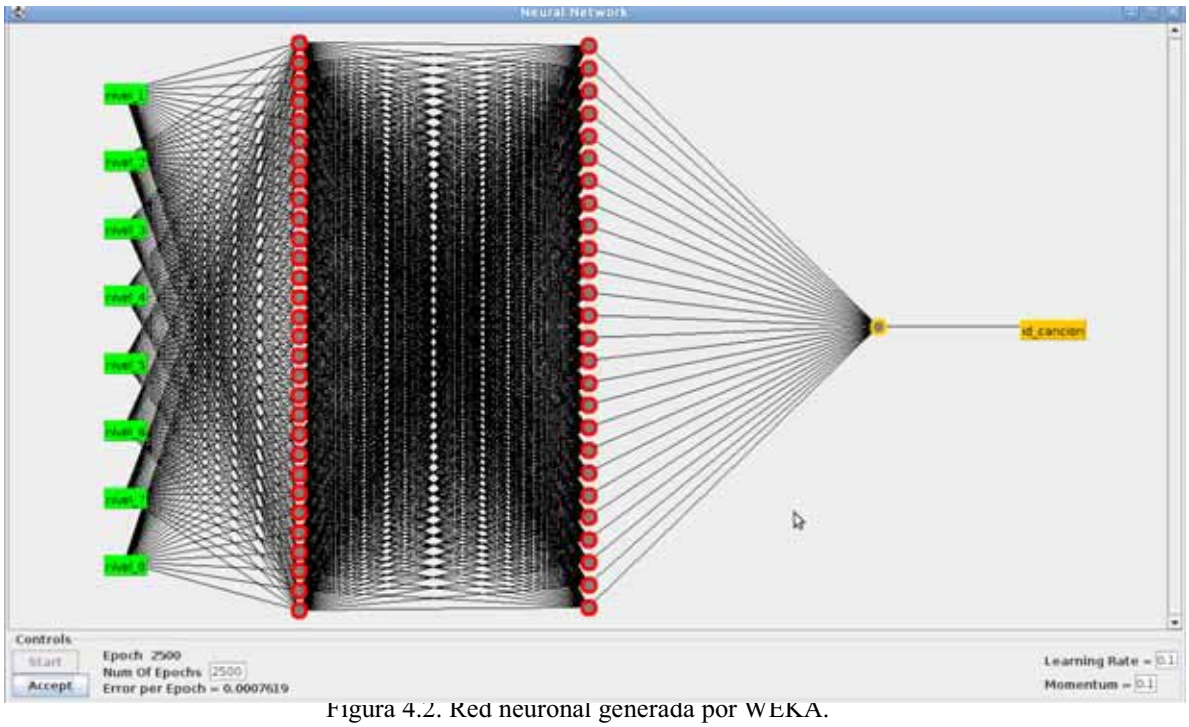

Figura 4.2. Red neuronal generada por WEKA.

2. Al finalizar el entrenamiento de la red neuronal se crea un archivo llamado ReporteWeka.txt el cual contiene el modelo de la red neuronal indicando los pesos correspondientes a cada conexión entre las neuronas, en seguida en otro archivo llamado PesosWeka.txt se almacenan únicamente el valor de los pesos que se encuentran el ReporteWeka.txt.

Ver código 3 en la sección códigos fuente. En el que se muestra el código completo correspondiente a esta fase.

Fase IV. Identificación de arquitectura de la red neuronal

1. Se realiza la interpretación de los pesos almacenados en el archivo PesosWeka.txt para tener una arquitectura correcta y lograr una buena generalización. Para esto se hicieron pruebas usando datos aleatorios del archivo de entrenamiento de la red neuronal obteniendo como salida una buena generalización. La lectura de los datos se realiza mediante un archivo en el que se simula, únicamente para esta fase y a modo de prueba, un tarareo. El nombre de este archivo es Tarareo.txt.

Ver código 4 en la sección códigos fuente. En el que se muestra el código completo correspondiente a esta fase.

5. Aplicación software

Se creó una aplicación software que permite grabar un tarareo en formato WAV monoaural de aproximadamente 11 segundos, suficientes para realizar el procesamiento de extracción de rasgos característicos y de esta manera crear un archivo de texto en el que se almacenan estos rasgos. Este archivo generado es el que se utiliza en la fase IV del capítulo anterior con la diferencia que ahora se trata de un tarareo y no de datos utilizados para el entrenamiento de la red neuronal.

Para la realización de esta aplicación fue necesario utilizar 3 clases implementadas en lenguaje JAVA, la primera de ellas se llama ventanaPrincipal.java (Ver código 5 en la sección códigos fuente), en la que se crea una interfaz gráfica usando la biblioteca de clase *swing*, y el *Sound API* de JAVA, la cual está integrada por 3 botones y un área en la que se mostraran los resultados de la consulta (ver figura 5.1). el botón Grabar realiza la grabación del tarareo del usuario creando un archivo llamado tarareo.wav dentro del directorio Tarareo.

Figura 5.1. Interfaz gráfica creada para mostrar los resultados de la consulta.

Una segunda clase usada para el funcionamiento de la aplicación es invocada desde la clase ventanaPrincipal.java al terminar la captura del tarareo y la creación del archivo tarareo.wav llamada analisisTarareo.java, en esta clase se lee el archivo tarareo.wav y se realiza el procedimiento descrito en el Capitulo 4 en la Fase I siguiendo los pasos 2, 3, 4, y 5, en este último paso no se realizan desplazamientos, al calcular la potencia por banda se mandan a normalizar los datos usando los valores obtenidos en la Fase II paso 1 del mismo capítulo almacenado los datos en un archivo de texto llamado Tarareo.txt

Ver código 5, 6 en la sección códigos fuente. En el que se muestra el código completo correspondiente a esta fase.

Por último esta clase invoca a la clase pesosWeka.java utilizada en la Fase IV del capítulo 4, la cual nos regresa una lista de 5 pistas de audio (tracks) ordenadas según su similitud con la consulta (tarareo). Al seleccionar una pista de audio tendremos la posibilidad de reproducirla presionando el botón Play y poder detenerla con el botón Stop, de esta manera poder realizar una nueva búsqueda o seleccionar otra pista de audio para su reproducción.

6. Conclusiones y trabajos futuros

En este trabajo se presento una metodología en la que se procesan archivos de sonido en formato WAV y a partir de este se realiza un reconocimiento de los mismos. Se observo que la transformada wavelet de Haar concentra la energía de una señal de audio obteniendo coeficientes con información tiempo-frecuencia que representan rasgos característicos de cada archivo de audio diferenciándolos entre sí. De esta manera también pudimos reducir el tamaño del vector representativo. Además fue suficiente el procesar únicamente la energía de cada banda de descomposición para poder identificar los archivos de sonido mediante el tarareo.

En trabajos futuros se sugiere hacer uso de otro tipo de wavelet como los Daubechies y de esta manera poder comparar los resultados obtenidos con el wavelet de Haar usado en este trabajo, poder incrementar en tiempo real el número de pistas almacenadas explorando y estudiando algoritmos que permitan un entrenamiento incremental, complementar este trabajo utilizando otro tipo de procesamiento para tener mejor precisión en la búsqueda y así poder procesar mayor cantidad de archivos de sonido.

Codigos fuente

1. Clase análisis.
   Clase que realiza la extracción de rasgos característicos de cada archivo de sonido almacenado en el directorio Canciones (10 en total) y los almacena en una base de datos

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.*;
import java.text.DecimalFormat;
import java.util.StringTokenizer;
/**
 *
 * @author Sergio Rivera
 */
public class analisis {

    public static String ChunkID = "";
    public static String Format = "";
    public static String Subchunk1ID = "";
    public static String Subchunk2ID = "";
    public static int ChunkSize,Subchunk2Size,Subchunk1Size;
    public static int AudioFormat,nChannels,SampleRate, ByteRate,BitsPerSample;
    public static int BlockAlign,nSamples;
    public static int NChunkSize,NSubchunk2Size,nmax,nc;
    public static double[] d16 ;
    public static double idCancion;
    public static short ventana;


public static void main(String[] args) {

        DataInputStream inFile = null;
        String nombreArchivo = "";
        String file ;
        File dir = new File("Canciones");
        String[] ficheros = dir.list();

        int x = 0;
            nc = ficheros.length;// # total de archivos de audio almacenados en Directorio

            try {
                if (ficheros == null)
                  System.out.println("No hay ficheros en el directorio especificado");
      else {
          while(x < nc){
              idCancion = 0;
              idCancion = (double)(x+1)/(nc+1);

              inicializar();                    // Inicializacion de las variables para cada cancion
              file =  ficheros[x];  // Nombre de cancion
              StringTokenizer st = new StringTokenizer(file, ".");
              nombreArchivo = st.nextToken();/* Nombre de archivo de audio a procesar*/
```

```
    file = dir + "/"+ file;
    inFile = new DataInputStream(new FileInputStream(file));
     System.out.println(nombreArchivo);
    // Obtención de información de la canción
     for (int i=0; i < 4; i++)
        ChunkID+=(char)inFile.readByte() ;
     for(int i=0; i < 4; i++)
        ChunkSize+= inFile.readUnsignedByte()*(int)Math.pow(256,i);

     for (int i= 0; i < 4; i++)
        Format+=(char)inFile.readByte() ;
     for (int i=0; i < 4; i++)
        Subchunk1ID+=(char)inFile.readByte() ;
     for(int i=0; i < 4; i++)
        Subchunk1Size+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 2; i++)
        AudioFormat+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 2; i++)
        nChannels+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 4; i++)
        SampleRate+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 4; i++)
        ByteRate+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 2; i++)
        BlockAlign+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
     for(int i=0; i < 2; i++)
        BitsPerSample+= inFile.readUnsignedByte();
     for (int i=0; i < 4; i++)
        Subchunk2ID+=(char)inFile.readByte() ;
     for(int i=0; i < 4; i++)
        Subchunk2Size+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
        nSamples = Subchunk2Size*8/(nChannels*BitsPerSample);
        ventana = (short) (Math.log(18*SampleRate)/Math.log(2.0));
        nmax = 1<<ventana;

      mostrarInformacion(); // Mostrar información de cada archivo wav
      // fin obtención de información

/***** Recuperación de información para procesamiento *****/
      short s =0;
      int j=0;
      short b16[]= new short[nSamples];
      double d[] = new double[nSamples];

     while (j < nSamples){
         s = 0;
         for(int i=0; i < 2; i++)
            s += inFile.readUnsignedByte()*(int)Math.pow(256,i);
            b16[j] = s;
            j++;
        }
/***** Fin recuperación de información    ***/


/*****    Codificacion Parson   **********/
```

```
                j=1;
                d[0] = 0.0;
                for (int i = 1; i < nSamples; i++) {
                   if(b16[i] == b16[i-1]){     //Repeat
                        d[j] = 0.5;
                    }
                   else if(b16[i] > b16[i-1]){ //Up
                        d[j] = 1.0;
                    }
                    else {                    //Down
                        d[j] = 0.0;
                     }
                      j++;
                 }
            /******* Fin Codificación Parson  *******/

                inFile.close();
                cortar( d ); // Procesamiento de infomación
                System.out.println("- Cancion "+nombreArchivo+" Procesada ----------\n");
                x++;
               }
                     }
                  } catch (IOException e) {
                                    e.printStackTrace();
                   }
          }

/*
 * Al invocar este método se realiza una conexión a la base de datos
 * tarareo en la que se almacenarán los resutados obtenidos despues de
 * calcular la potencia por nivel de transformacion de wavelet Haar en
 * la tabala potencias
 *
 * */
   public static void cortar( double[]b16 ){
        int secs = 40,i = 0;
        int niveles = 7;
        d16 = new double[nmax];
        String url = "jdbc:postgresql://localhost:5432/tarareo";
        String user = "postgres";
        String password = "204243";
        double nPotencia [] = new double[9];
        DecimalFormat df = new DecimalFormat("0.00");
        /* Conexion BD  */
        try{
           Class.forName("org.postgresql.Driver");
           } catch (ClassNotFoundException cnfe){
             System.out.println("No se pudo encontrar el controlador JDBC");
             System.exit(1);
           }
        Connection conexion = null;
        try {
           conexion = DriverManager.getConnection(url, user, password);

           Statement s = conexion.createStatement();
```

```
                for(int  off=0;  off+nmax<SampleRate*secs&&off+nmax<nSamples;off+=SampleRate/30){

                        for(i = 0; i < nmax; i++){// Corte de bloques de 2 ¹⁶
                                d16[i]=b16[i+off];
                        }
                        for(i = 0; i <= niveles;i++){
                                //Aplicación de la transformada Haar
                                transformHaar(nmax>>i);
                                /** Calculo de Potencia por nivel de transformación
                                 * almacenandola en nPotencia*/
                                nPotencia [i] = potencia(d16,(nmax>>i)/2);
                        }

                    nPotencia [niveles+1] = idCancion;/*Asignacion de id para cada archivo de audio*/

          //Guardar en BD tarareo en tabala potencias vector nPotencia
                s.execute("insert  into  potencias  values('"+(int)nPotencia[0]+"','"+(int)nPotencia[1]+"','"  +

                                        "'"+(int)nPotencia[2]+"','"+(int)nPotencia[3]+"'," +

                                        "'"+(int)nPotencia[4]+"','"+(int)nPotencia[5]+"'," +

                                        "'"+(int)nPotencia[6]+"','"+(int)nPotencia[7]+"'," +

                                        "'"+df.format(nPotencia[8])+"')");



                    }
                s.close();
                conexion.close();
          } catch (SQLException sqle) {
            System.out.println("No se pudo establecer conexión");
            System.exit(1);
          }
        }


/*
 * Metodo que realiza la transformada wavelet Haar aplicada a cada n cantidad de datos
 *
 * */
   private static void transformHaar(int n) {
      int i=0, j;
      double tmp[] = new double[n/2];

       for (j = 0; j < n; j = (j + 2)) {
         tmp[i] = (d16[j] + d16[j+1])/2;
         i++;
         }
       for (i = 0; i < n/2; i++) {
         d16[i] = tmp[i];
         }
   }
```

```
/*
 * Método que realiza el cálculo de la potencia almacenada en b
 * regresando la suma total del cuadrado de cada valor en b
 *
 * */

private static double potencia(double[] b, int max) {

     double suma = 0;

     for(int j=0; j < max;j++){
       suma += Math.pow(b[j], 2);
     }

     return suma;
}


/*
 * Inicialización de las variables globales para la lectura de
 * la cabecera de cada archivo de audio WAV
 * */

private static void inicializar() {
    ChunkID = "";
    Format = "";
    Subchunk1ID = "";
    Subchunk2ID = "";
    ChunkSize = 0;
    Subchunk2Size=0;
    Subchunk1Size=0;
    AudioFormat=0;
    nChannels=0;
    SampleRate=0;
    ByteRate=0;
    BitsPerSample=0;
    BlockAlign=0;
    nSamples=0;
    NChunkSize=0;
    NSubchunk2Size=0;
    nmax=0;
}
```

```
/*
    * Método que imprime en pantalla la información de la cabecera de
    *cada archivo de audio a porcesar
    *
    * */

  private static void mostrarInformacion() {
        System.out.println("ChunkID:         "+ChunkID);
        System.out.println("ChunkSize:       "+ChunkSize);
        System.out.println("Format:          "+Format);
        System.out.println("Subchunk1ID:     "+Subchunk1ID);
        System.out.println("Subchunk1Size:   "+Subchunk1Size);
        System.out.println("Canales:         "+nChannels);
        System.out.println("SampleRate:      "+SampleRate);
        System.out.println("BlockAlign:      "+BlockAlign);
        System.out.println("BitsPerSample:   "+BitsPerSample);
        System.out.println("Subchunk2ID:     "+Subchunk2ID);
        System.out.println("Subchunk2Size:   "+Subchunk2Size);
        System.out.println("nSamples:        "+nSamples);
        System.out.println("\nLa ventana es de 2^"+ventana+" = "+nmax+"\nequivalente a
"+(float)nmax/SampleRate+" segundos");
        System.out.println();
    }
}
```

2. Clase recuperacionDatos.
   Clase que crea el archivo train.arff leyendo los datos almacenados en la tabla potencias de la base de datos tarareo normalizando cada nivel de portencia

```
Import java.io.FileWriter;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;


public class recuperacionDatos {

        public static int []max= new int[8];
   public static int []min= new int[8];

        public static void main(String[] args) {

                double x[] = new double[8];
                String url = "jdbc:postgresql://localhost:5432/tarareo";
      String user = "postgres";
      String password = "204243";
      Statement s;
      ResultSet rs;
      String consulta=null;

      double []nivel = new double[9];/* arreglo en que se almacenarán los datos
                                        leidos desde la base de datos tarareo*/

      Connection conn = null;
                try{
        Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException cnfe){
          System.out.println("No se pudo encontrar el controlador JDBC");
          System.exit(1);
        }

      try {
         conn = DriverManager.getConnection
                (url, user, password);

        int i;
        s = conn.createStatement();
        for ( i = 1; i<=8;i++){
            consulta="";
          consulta = "SELECT max(nivel_"+i+") FROM potencias";
          rs = s.executeQuery(consulta);
                while (rs.next())
                {
                max[i-1] =rs.getInt(1);/*Almacenamiento del valor maximo en variable
                                        max                     */
                }
```

```
            consulta="";
            consulta = "SELECT min(nivel_"+i+") FROM potencias";
            rs = s.executeQuery(consulta);
                        while (rs.next())
                        {
                                    min[i-1] =rs.getInt(1);  /*Almacenamiento del valor mínimo en variable
                                                                    min                 */
                        }
            }

          crearCabecera();  // crear cabecera de archivo con formato arff


            consulta="";
            consulta = "SELECT * FROM potencias";
            rs = s.executeQuery(consulta);
            /*
             * Lectura de los datos almacenados en la base de datos
             *                                                          */
                        while (rs.next()){
                                    for ( i = 1; i<=8;i++){
                                                nivel[i-1] = rs.getInt(i);
                                                }
                                    nivel[8] = rs.getDouble(9);//almacenamiento del identificador del archivo de sonido

                                    normalizar(nivel); //Normalización de los datos leídos desde la base de datos
                        }
           s.close();
           conn.close();
       } catch (SQLException sqle) {
           System.out.println("Conexión fallida");
           System.exit(1);
         }
}
/*
 * Método que crea archivo trainWeka.arff junto con la cabecera
 *
 * */
private static void crearCabecera() {
            FileWriter fichero = null;
     PrintWriter pw = null;

     try
       {
        fichero = new FileWriter("trainWeka.arff");
        pw = new PrintWriter(fichero);
           pw.println("@relation trainWeka\n");
           pw.println("@attribute nivel_1 NUMERIC");
           pw.println("@attribute nivel_2 NUMERIC");
           pw.println("@attribute nivel_3 NUMERIC");
           pw.println("@attribute nivel_4 NUMERIC");
           pw.println("@attribute nivel_5 NUMERIC");
           pw.println("@attribute nivel_6 NUMERIC");
           pw.println("@attribute nivel_7 NUMERIC");
           pw.println("@attribute nivel_8 NUMERIC");
```

```
            pw.println("@attribute id_cancion NUMERIC");
            pw.println("@data\n");
          fichero.close();
        } catch (Exception e) {
          e.printStackTrace();
       }
}

/*Método que normaliza los datos almacenados en la variable x
 * y manda a gurdarlos en el archivo trainWeka.arff
 * */
private static void normalizar(double[] x) {

            double resultados[] = new double [9];

            for(int i =0; i < (x.length)-1; i++){
                        resultados[i] = (x[i] - min[i])/(max[i] - min[i]);
            }
              resultados[8] = x[8];
            // guarda resulatados
              guarda(resultados);

}

/*
 * Método que agrega al final del archivo trainWeka.arff los datos normalizados
 * */
private static void guarda(double[] b) {

        FileWriter fichero = null;
        PrintWriter pw = null;

        try
          {
            fichero = new FileWriter("trainWeka.arff",true);
            pw = new PrintWriter(fichero);
              for(int j=0; j < (b.length)-1;j++){

                pw.print(b[j]+",");
              }
              pw.println(b[8]);
            fichero.close();
         } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3. Clase perceptron.
   Clase que implementa la red neuronal (Perceptron Multilayer) que lee el archivo trainWeka.arff para su entrenamiento y genera un archivo llamado ReporteWeka.txt en el que se muestra la arquitectura de la red neuronal indicando los pesos y las conexiones entre neuronas (nodos). También genera un archivo llamado PesosWeka.txt en el que ‘únicamente se almacenan los pesos a partir del archivo ReporteWeka.txt.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.StringTokenizer;
import weka.core.Instances;
import weka.core.Utils;
import weka.classifiers.Evaluation;
import weka.classifiers.functions.MultilayerPerceptron;

public class perceptron {

        /**
         * Opciones para definir una arquitectura con WEKA

        -L. Define la tasa de aprendizaje (default 0.3)

        -M. Define el momentum (default 0.2)

        -N. Especifica el número de epocas para parar el entrenamiento (default 500)

        -V  default 0. No se realiza validación, mientras se especifique un número de épocas

        -S Establecer la semilla para el generador de números aleatorios. (default 0)

        -E Ajuste del umbral para el número de errores consecutivos permitidos durante la prueba de
        validación. (default 20)

        -G Permite visualizar la arquitectura de la red neuronal.

        -B No preprocesa los casos automáticamente usando un valor nominal de filtro binario .

        -H Establece el numero de nodos que se utilizarán en la arquitectura. Cada numero deberá ir
        separado por comas.(default 4)

        -C No normaliza los datos de entrenamiento

        -I No normaliza los datos de los atributos.

        -R No permite a la red reseteo automatico
         *
         */
```

```
        public static void main(String[] args) throws Exception {
try{
          FileReader trainreader = new FileReader("trainWeka.arff");
       Instances train = new Instances(trainreader);
          train.setClassIndex(train.numAttributes() - 1);

      MultilayerPerceptron mlp = new MultilayerPerceptron();//
        mlp.setOptions(Utils.splitOptions("-L 0.1 -M 0.1 -N 2500 -V 0 -S 0 -E 1 -H 30,26 - G -B -C -I -R"));

                  mlp.buildClassifier(train);

                  System.out.println(mlp);// Muestra la arquitectura indicando los pesos y las conexiones
                                          // entre neuronas (nodos)

                 Evaluation eval = new Evaluation(train);
                  eval.evaluateModel(mlp, train);// Evaluación del entrenamiento


                  System.out.println(eval.toSummaryString("\nResultados\n", false));

                   //Creación de archivo en el que se almacenan los resultados
        BufferedWriter writer = new BufferedWriter(new FileWriter("ReporteWeka.txt"));
                  writer.write(""+mlp);
                  writer.flush();
                  writer.close();
                  trainreader.close();


        StringTokenizer tokens;
        FileReader fr = null;
        FileWriter fw = null;
        PrintWriter pw = null;

        fw = new FileWriter("PesosWeka.txt");
        pw = new PrintWriter(fw);
          //Extracción de los pesos
             fr = new FileReader("ReporteWeka.txt");
             BufferedReader bf = new BufferedReader(fr);
             String linea;
             String palabra;
             double num = 0.0,d=0.0;
             int i;

             while((linea = bf.readLine())!= null){
                  tokens = new StringTokenizer(linea," ");
                  while(tokens.hasMoreTokens()){
                            palabra = tokens.nextToken();

                            if( verifica(palabra) == true){
                                      num = Double.parseDouble(palabra);
                                      i = (int)num;
                                      d = num - i;
                                      if (d != 0.0)
                                                pw.println(num);
```

```
                                        }
                                    }
                                }
                                pw.close();
                                fw.close();
                                fr.close();

                                System.out.println("Archivo Creado Exitosamente");
                                    } catch(Exception ex){
                                            ex.printStackTrace();
                                    }
        }
/*
*Método que verifica si el caracter leído es número o no, de ser numero regresa true
*/
        private static boolean verifica(String palabra) {
                double num = 0.0;
                try{
                        num = Double.parseDouble(palabra);
                }catch (Exception e) {
                        return false;
                }
                return true;
        }
}
```

4. Clase pesosWeka

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.StringTokenizer;
import java.util.logging.Level;
import java.util.logging.Logger;


public class pesosWeka {
        public static int intput = 9;//8 entradas + BIAS
        public static int nC1 = 30;// # neuronas de la capa 1 30
        public static int nC2 = 26;// # neuronas de la capa 2 26
        public static double [][] wij = new double [nC1+1][intput+1] ;// Almacenamiento de Pesos de
       //Entrada a Capa 1
        public static double [][] wjk = new double [nC2+1][nC1+1] ; // Almacenamiento de Pesos de Pesos
       //de Capa 1 a Capa 2
        public static double [] s = new double [nC2+1]; // Salida
        public static double [] in = new double [intput];// Datos de entrada
        public static double [] sumasC1 = new double [nC1+1];// Calculo tanh en C1
        public static double [] sumasC2 = new double [nC2+1];// Calculo sigmoidal en C2

        public static int recupera() throws IOException{
                        cargarPesos();
                                System.out.println("Pesos Cargados");
                        leerTarareo();
                System.out.println("Tarareo leido");
            int i = recuperarCancion();
          return i;
        }
/*
 *Metodo que regresa el valor del identificador del archive de sonido asociado
 */
private static int recuperarCancion() {

                DecimalFormat df = new DecimalFormat("0.00");
          double r, entero, decimal;
                sumatoriaC1();//Sumatoria cada peso por entrada
                sigmoidalC1();
                sumatoriaC2();
                sigmoidalC2();


                r = salida();

             System.out.println("\nsalida red: "+df.format(r));
              r = (r*11); // Recuperar cancion
              entero=(long)r;
              decimal=r-entero;

             if(decimal >= 0.5){
```

```
                    entero = (int)(entero+1);
                }
                if(entero >= 10) entero = 9.0;
                    return (int)entero%11;
            }

/*
 * Método que realiza la función sigmoidal a cada sumatoria
 * correspondiente a cada neurona de la Capa 2
 */
private static void sigmoidalC2() {
                    double e = 0.0;
        for (int i = 1; i<= nC2; i++){
            e = Math.exp(-(sumasC2[i]));
            sumasC2[i] = (1/(1+e));
            }

             }

             private static double salida() {
                    double suma = 0.0;
                for (int l = 0; l <= nC2; l++){
                    suma += sumasC2[l]*s[l];
                    }
                return suma;
            }
/*
 * Método que realiza la sumatoria de la multiplicación de cada entrada con
 * su respectivo peso correspondiente a la Capa 2
 */
private static void sumatoriaC2() {
                    double suma = 0.0;
                 // Sumatoria de datos de entrada por su respectivo peso
                   sumasC2[0]=1; // Dato correspondiente al Bias = 1
                   for (int k = 1; k<= nC2; k++){
                      for (int j=0;j<=nC1; j++){
                         suma += sumasC2[j]*wjk[k][j];
                       }
                       sumasC2[k] = suma;
                       suma = 0.0;
                }// fin sumatoria

            }
/*
 * Método que realiza la función sigmoidal a cada sumatoria
 * correspondiente a cada neurona de la Capa 1
 */
private static void sigmoidalC1() {
                    double e = 0.0;
        for (int i = 1; i<= nC1; i++){
            e = Math.exp(-(sumasC1[i]));
            sumasC1[i] = (1/(1+e));
            }

             }
```

```
/*
 * Método que realiza la sumatoria de la multiplicación de cada entrada con
 * su respectivo peso correspondiente a la Capa 1
 */
private static void sumatoriaC1() {
                    double suma = 0.0;
                System.out.println();
                // Sumatoria de datos de entrada por su respectivo peso
                sumasC1[0]=1;///bias
                  for (int j = 1; j<= nC1; j++){
                    for (int i=0;i<intput; i++){

                     suma += in[i]*wij[j][i];
                     }
                     sumasC1[j] = suma;
                     suma = 0.0;
                   }// fin sumatoria

        }
/*
 * Método que lee el archivo tarareo.txt
 */
private static void leerTarareo()throws IOException {
                      try {
                    int i=1;
                    FileReader fr = null;
                    fr = new FileReader("tarareo.txt");
                    BufferedReader bf = new BufferedReader(fr);
                    String sCadena;
                    sCadena = bf.readLine();
                    StringTokenizer st = new StringTokenizer(sCadena);
                       in[0]=1; // Bias
                      while(st.hasMoreTokens()){
                          String str = st.nextToken(",");
                          in[i] = Double.parseDouble(str);
                          i++;
                      }
                } catch (FileNotFoundException ex) {
                    Logger.getLogger(pesosWeka.class.getName()).log(Level.SEVERE, null, ex);
                }

        }
/*
 * Método que realiza la lectura del archivo PesosWeka.txt cargando los pesos asociados
 * en cada conexión entre neuronas presentado en la arquitectura de la red neuronal.
 *  la lectura está basada al formato de salida que presenta WEKA.
 */

private static void cargarPesos() throws NumberFormatException, IOException {
                    double n =0;
      int i =1,cont=1,linea;
      int j = 1, k = 1, l = 1;
      FileReader fr = null;
      fr = new FileReader("PesosWeka.txt");
      BufferedReader bf = new BufferedReader(fr);
```

```
        //   Lectura de los pesos asociados al Bias a las diferentes capas
             for (linea = 0; linea <= (nC2); linea ++){
                s[linea] = Double.parseDouble(bf.readLine());
                }

             //Se cargan los pesos correspondientes a cada entrada en wij
             for (i = 1 ; i<=nC1; i++){
                for(j = 0; j<intput;j++){
                   wij[i][j] = Double.parseDouble(bf.readLine());
                }
             }
    // fin de carga de pesos

          //Se cargan los pesos correspondientes a cada entrada en wjk
             for (j =1; j<=nC2; j++){
                for(k = 0; k<=nC1;k++){
                      wjk[j][k] = Double.parseDouble(bf.readLine());

                }
             }

          fr.close();
              }

}
```

5. Clase ventanPrincipal.
   Clase que implementa la interfaz gráfica que permite grabar un tarareo por aproximadamente 11 segundos para obtener una lista con las 5 pistas de sonido con mas similitud al tarareo y que permite reproducir la pista seleccionada de la lista presionado el botón Play y detener la reproducción de la misma al presionar el botón Detener y de esta manera poder realizar otra consulta.

```
import javax.swing.SwingUtilities;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.Rectangle;
import java.io.*;
import javax.sound.sampled.*;
import java.util.Timer;
import java.util.TimerTask;
import java.awt.Color;
import javax.swing.JList;

public class ventanaPrincipal {
	Timer timer;
	int segundos;
	AudioFormat formatoAudio;
	TargetDataLine targetDataLine;
	private JFrame jFrame = null;
	private JPanel jContentPane = null;
	private JButton BotonGrabar = null;
	private JList jList = null;
	private JButton Play = null;
	private JButton botonPlay = null;
	private JButton botonStop = null;
	private Clip audio;
	private AudioFileFormat aff;
	private AudioInputStream ais;
	/**
	 * This method initializes jFrame
	 *
	 * @return javax.swing.JFrame
	 */
	private JFrame getJFrame() {
		if (jFrame == null) {
			jFrame = new JFrame();
			jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			jFrame.setSize(536, 331);
			jFrame.setBackground(new Color(185, 189, 235));
			jFrame.setContentPane(getJContentPane());
			jFrame.setTitle("Proyecto Terminal Tarareo");
		}
		return jFrame;
	}
```

```
/**
 * Este meetodo inicializa el JPanel
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane() {
                        if (jContentPane == null) {
                                jContentPane = new JPanel();
                                jContentPane.setLayout(null);
                                jContentPane.setBackground(new Color(209, 220, 233));
                                jContentPane.add(getBotonGrabar(), null);
                                jContentPane.add(getJList(), null);

                                jContentPane.add(getBotonPlay(), null);
                                jContentPane.add(getBotonStop(), null);
                        }
                        return jContentPane;
                }

/**
 * Este metodo inicializa el JButton
 *
 * @return javax.swing.JButton
 */
private JButton getBotonGrabar() {
                        if (BotonGrabar == null) {
                                BotonGrabar = new JButton();
                                BotonGrabar.setBounds(new Rectangle(61, 28, 105, 30));
                                BotonGrabar.setText("Grabar");
                                BotonGrabar.addActionListener(new java.awt.event.ActionListener() {
                                        public void actionPerformed(java.awt.event.ActionEvent e) {

                                                BotonGrabar.setText("Grabando");
                                                BotonGrabar.setEnabled(false);
                                                segundos = 0;

                                                CapturarAudio();
                                                detener(13);

                                        }
                                });
                        }
                        return BotonGrabar;
                }

// Metodo que captura el audio desde el microfono y lo salva en una archivo
// tipo wav
 private void CapturarAudio(){
                try{
                  //Get things set up for capture
                  formatoAudio = getAudioFormat();
                  DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class,
                                formatoAudio);
                     targetDataLine = (TargetDataLine)
                       AudioSystem.getLine(dataLineInfo);
```

```
                //Creación de un thread para capturar desde el microfono
                // El cual estara en ejecución hasta que pasen 11 segundos
                // aproximadamente
                new CaptureThread().start();
               }catch (Exception e) {
                e.printStackTrace();
                System.exit(0);
               }
  }

/*
 * Método que detendra la captura de audio a los 11 segundos de inicio de la captura
 * */
 private void detener(long segundos){
            timer = new Timer();

            timer.schedule(new tarea(), segundos*1000);

    }
    class tarea extends TimerTask {
        public void run() {
            System.out.println ( "Listo" ) ;
            timer.cancel(); //detenemos el timer
            targetDataLine.stop();
               targetDataLine.close();
               BotonGrabar.setEnabled(true);
               BotonGrabar.setText("Grabar");
             /* Se invoca al método pausar de la clase analisisTarareo que permite
              * la finalización de la creación del archivo tarareo.wav*/
               analisisTarareo.pausar();
               String []lista;
                                try {
                                        /*Invocación al metodo analizaTarareo de la calse analisisTarareo
                                         * la cual se encarga de regresar una lista de pistas que posteriormente
                                         * se presentan en la aplicación*/
                                        lista = analisisTarareo.analizaTarareo();
                                        jList.setListData(lista);
                                        botonPlay.setEnabled(true);
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
        }
    }
```

```
/*
 * Método que extrae la cabecera del archivo a generar
 * */
 private AudioFormat getAudioFormat(){

                    float sampleRate = 6000.0F;
              int sampleSizeInBits = 16;
              int channels = 1;
              boolean signed = true;
              boolean bigEndian = false;
              return new AudioFormat(sampleRate,
                                  sampleSizeInBits,
                                  channels,
                                  signed,
                                  bigEndian);
            }

 /*
  * Método que crea el archivo de sonido que contiene
  * el tarareo del usuario
  *
  * */
          class CaptureThread extends Thread{
            public void run(){
              AudioFileFormat.Type tipoArchivo = null;
              File audioFile  = null;
              tipoArchivo     = AudioFileFormat.Type.WAVE;
              audioFile       = new File("Tarareo/tarareo.wav");
              try{
                targetDataLine.open(formatoAudio);
                targetDataLine.start();
                AudioSystem.write(new AudioInputStream(targetDataLine),
                                tipoArchivo,audioFile);

              }catch (Exception e){
                e.printStackTrace();

              }
            }
          }

/**
 * este metodo  inicializa el JList
 *
 * @return javax.swing.JList
 */
          private JList getJList() {
                    if (jList == null) {
                              jList = new JList();
                              jList.setBounds(new Rectangle(45, 106, 306, 142));

                    }
                    return jList;
          }
```

```
/**
 * Este método inicializa el botonPlay
 */
private JButton getBotonPlay() {
		if (botonPlay == null) {
			botonPlay = new JButton();
			botonPlay.setBounds(new Rectangle(361, 105, 83, 28));
			botonPlay.setText("Play");
			botonPlay.setEnabled(false);


			botonPlay.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent e) {
					try {
						botonPlay.setEnabled(false);
						botonStop.setEnabled(true);
						BotonGrabar.setEnabled(false);

						playAudio();

					} catch (LineUnavailableException e1) {
						// TODO Auto-generated catch block
						e1.printStackTrace();
					} catch (IOException e1) {
						// TODO Auto-generated catch block
						e1.printStackTrace();
					}
				}
			});
		}
	 return botonPlay;
	}
/**
 * Este método inicializa el botonStop
 * y es usado para detener la reproducción de
 *	la pista de audio
 */
private JButton getBotonStop() {
		if (botonStop == null) {
			botonStop = new JButton();
			botonStop.setBounds(new Rectangle(364, 150, 80, 26));
			botonStop.setText("Stop");
			botonStop.setEnabled(false);
			botonStop.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("Parando la reproducción");
					botonPlay.setEnabled(true);
					BotonGrabar.setEnabled(true);
					botonStop.setEnabled(false);
					audio.stop();
				}
			});
		}
	return botonStop;
}
```

```
/*
 * Método que realiza la reproducción de la pista de audio
 * */

public void playAudio() throws LineUnavailableException, IOException{

                    try{
                        String archivo = null;
                            archivo = "Canciones/"+(String)jList.getSelectedValue();
                            aff = AudioSystem.getAudioFileFormat(new File(archivo));
                            ais = AudioSystem.getAudioInputStream(new File(archivo));

                            AudioFormat af = aff.getFormat();
                            DataLine.Info info=new DataLine.Info(Clip.class,ais.getFormat(),
                                                    ((int) ais.getFrameLength()*af.getFrameSize()));

                            audio = (Clip)AudioSystem.getLine(info);
                            audio.open(ais);
                            audio.loop(0);
                    }catch(UnsupportedAudioFileException ee){
                            System.out.println("Formato de archivo no soportado");
                    }
}

/**
 * Método principal que lanza la aplicación
 */
public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                        ventanaPrincipal application = new ventanaPrincipal();
                        application.getJFrame().setVisible(true);
                }
        });
    }
}
```

6. Clase analisisTarareo.
   Clase que realiza la extracción de los rasgos característicos del tarareo y los almacena en el archivo tarareo.txt

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class analisisTarareo {
        public static String ChunkID = "";
   public static String Format = "";
   public static String Subchunk1ID = "";
   public static String Subchunk2ID = "";
   public static int ChunkSize,Subchunk2Size,Subchunk1Size;
   public static int AudioFormat,nChannels,SampleRate, ByteRate,BitsPerSample;
   public static int BlockAlign,nSamples;
   public static int NChunkSize,NSubchunk2Size,nmax,nc;
   public static short ventana;
   public static double[] d16 ;

        public static String [] analizaTarareo() throws IOException{


         short s =0;
    int j=0;
    short b16[];
    double d[] ;
    int id = 0;
    File dir = new File("Canciones");
         String[] canciones = dir.list();
        String [] contenido = new String[5] ;

         try {
            DataInputStream inFile = new DataInputStream(newileInputStream("Tarareo/tarareo.wav"));

                inicializar();
                // Obtención de información de la canción
                for (int i=0; i < 4; i++)
                        ChunkID+=(char)inFile.readByte() ;
                for(int i=0; i < 4; i++)
                        ChunkSize+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for (int i= 0; i < 4; i++)
                        Format+=(char)inFile.readByte() ;
                for (int i=0; i < 4; i++)
                        Subchunk1ID+=(char)inFile.readByte() ;
                for(int i=0; i < 4; i++)
                        Subchunk1Size+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for(int i=0; i < 2; i++)
                        AudioFormat+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
```

```
            for(int i=0; i < 2; i++)
                        nChannels+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for(int i=0; i < 4; i++)
                        SampleRate+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for(int i=0; i < 4; i++)
                        ByteRate+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for(int i=0; i < 2; i++)
                        BlockAlign+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                for(int i=0; i < 2; i++)
                        BitsPerSample+= inFile.readUnsignedByte();
                for (int i=0; i < 4; i++)
                        Subchunk2ID+=(char)inFile.readByte() ;
                for(int i=0; i < 4; i++)
                        Subchunk2Size+= inFile.readUnsignedByte()*(int)Math.pow(256,i);
                        nSamples = Subchunk2Size*8/(nChannels*BitsPerSample);
                        ventana = (short) (Math.log(18*SampleRate)/Math.log(2.0));
                        nmax = 1<<ventana;

    mostrarInformacion(); // Mostrar información de cada archivo wav

   // Recuperación de información para procesamiento
      b16 = new short[nSamples];
      d   = new double[nSamples];

     while (j < nSamples){
         s = 0;
         for(int i=0; i < 2; i++)
            s += inFile.readUnsignedByte()*(int)Math.pow(256,i);
            b16[j] = s;
            j++;
        }
    // Fin recuperación de información

// *****    Codificacion Parson   **********
      j=1;
      d[0] = 0.0;
      for (int i = 1; i < nSamples; i++) {
         if(b16[i] == b16[i-1]){     //Repeat
               d[j] = 0.5;
          }
         else if(b16[i] > b16[i-1]){ //Up
               d[j] = 1.0;
          }
          else {                    //Down
               d[j] = 0.0;
           }
            j++;
       }
 // ******* Fin Codificación Parson  *******

     inFile.close();

     cortar(d); // Procesamiento de infomación

     id = pesosWeka.recupera(); //Recupera el identificador del archivo de sonido
```

```
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		System.out.println("Id canción: "+id);
		if (id <= 2 ){

			contenido [0]= ""+canciones[id-1];
			contenido [1]= ""+canciones[id];
			contenido [2]= ""+canciones[id+1];
			contenido [3]= ""+canciones[(id+9)%10];
			contenido [4]= ""+canciones[(id+8)%10];
		}
		else {
			contenido [0]= ""+canciones[id-1];
			contenido [1]= ""+canciones[id%10];
			contenido [2]= ""+canciones[(id+1)%10];
			contenido [3]= ""+canciones[(id-2)%10];
			contenido [4]= ""+canciones[(id-3)%10];
		}
		return contenido;

		}

/*
*Realiza la extracción de rasgos característicos del tarareo y los manda a normalizar
*/
private static void cortar(double[] b16) {
			int i = 0;
	int niveles = 7;
	d16 = new double[nmax];
	double potenciaBanda[] = new double[8];

	   for(i = 0; i < nmax; i++){// Corte de bloques de 2¹7
	      d16[i]=b16[i];
	      }

	   for(i = 0; i <= niveles;i++){
	     //Aplicación de la transformada Haar
	     potenciaBanda[i] = transformHaar(nmax>>i);

	      }
	   normalizar(potenciaBanda);

	   }

/*
*Normaliza el vector x y manda a guardar los datos a un archive de texto
*/
private static void normalizar(double[] x) {
			int min[] = {11636,4901,2186,1032,499,247,123,61};
			int max[] = {13910,6060,2629,1191,543,268,134,66};
			double resultados[] = new double [8];
```

```
                    for(int i =0; i < x.length; i++){
                            resultados[i] = (x[i] - min[i])/(max[i] - min[i]);
                    }

                    // guarda Potencia por banda almacenada en d16, cada nmax>>i datos
                       guarda(resultados);
            }
 /*
  * Método que realiza el cálculo de la transformada wavelet Harr y
  * calcula la potencia aplicada a cada nivel de transformación
  * regresando la suma total del cuadrado de cada valor en b
  *
  * */
private static double transformHaar(int n) {
                int i=0, j;
                double tmp[] = new double[n/2];
                double suma = 0;
                 for (j = 0; j < n; j = (j + 2)) {
                      tmp[i] = (d16[j] + d16[j+1])/2;
                   i++;
                   }
                 for (i = 0; i < n/2; i++) {
                       d16[i] = tmp[i];
                      suma += Math.pow(tmp[i], 2);
                      }
                 return suma;
            }

/*
   * Método que gurda los datos en archivo de texto tarareo.txt
   *
   * */
    private static void guarda(double[] b) {
        FileWriter fichero = null;
        PrintWriter pw = null;

        try
          {
            fichero = new FileWriter("tarareo.txt");
            pw = new PrintWriter(fichero);
              for(int j=0; j < b.length;j++){

                pw.print(b[j]+",");
              }

              fichero.close();
         } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
 /*
   * Método suspende la ejecución del programa por 3 segundos permitiendo
   * la finalización de creación del archivo tarareo.wav
   *
  * */
public static void pausar () {
                        try {
                        Thread.sleep (3000);
                        } catch (InterruptedException e) {

                        }
            }
 /*
   * Método que muestra la cabecera de cada archivo de sonido a procesar
   * */
private static void mostrarInformacion() {
       System.out.println("ChunkID:         "+ChunkID);
       System.out.println("ChunkSize:       "+ChunkSize);
       System.out.println("Format:          "+Format);
       System.out.println("Subchunk1ID:     "+Subchunk1ID);
       System.out.println("Subchunk1Size:   "+Subchunk1Size);
       System.out.println("Canales:         "+nChannels);
       System.out.println("SampleRate:      "+SampleRate);
       System.out.println("BlockAlign:      "+BlockAlign);
       System.out.println("BitsPerSample:   "+BitsPerSample);
       System.out.println("Subchunk2ID:     "+Subchunk2ID);
       System.out.println("Subchunk2Size:   "+Subchunk2Size);
       System.out.println("nSamples:        "+nSamples);
      System.out.println("\nLa ventana es de 2^"+ventana+" = "+nmax+"\nequivalente a
        "+(float)nmax/SampleRate+" segundos");
       System.out.println();
    }
 /*
   * Método que inicializa las variables de la cabecera para cada archivo de sonido a procesar
   * */
private static void inicializar() {
       ChunkID = "";
       Format = "";
       Subchunk1ID = "";
       Subchunk2ID = "";
       ChunkSize = 0;
       Subchunk2Size=0;
       Subchunk1Size=0;
       AudioFormat=0;
       nChannels=0;
       SampleRate=0;
       ByteRate=0;
       BitsPerSample=0;
       BlockAlign=0;
       nSamples=0;
       NChunkSize=0;
       NSubchunk2Size=0;
       nmax=0;
    }
}
```

Ejecución del Software

Requisitos

- Tener instalada la maquina virtual de java JRE V. 6.
  (Sino cuenta con JRE instalada en su equipo dirigirse a la siguiente liga http://www.java.com/es/ en donde encontrara la versión adecuada así como las instrucciones de instalación para diferentes sistemas operativos).

Ejecución.

Para usuarios Linux copiar la carpeta tarareo en el escritorio, después abrir una consola y ubicarse en la ruta en donde se encuentra el directorio tarareo, una vez dentro del directorio ejecutar el siguiente comando:

java –jar tarareo.jar

Para usuarios Windows, simplemente ubicarse en la carpeta tarareo y hacer doble click en el archivo tarareo.jar

Creación de Base de Datos

Para la realización de este trabajo se utilizo el manejador PostgreSQL 8.4 creando una base de datos llamada tarareo y una tabla llamada potencias. A continuación se muestran las sentencias SQL para la creación tanto para la base de datos como para la tabla.

Sentencias SQL para la creación de base datos tarareo

```
CREATE DATABASE tarareo
  WITH OWNER = postgres
       ENCODING = 'UTF8'
       TABLESPACE = pg_default
       LC_COLLATE = 'es_MX.UTF-8'
       LC_CTYPE = 'es_MX.UTF-8'
       CONNECTION LIMIT = -1;
```

.

Sentencias SQL para la creación de tabla potencias en base datos tarareo.

```
CREATE TABLE potencias
(
  nivel_1 integer,
  nivel_2 integer,
  nivel_3 integer,
  nivel_4 integer,
  nivel_5 integer,
  nivel_6 integer,
  nivel_7 integer,
  nivel_8 integer,
  id_cancion double precision
)
WITH (
  OIDS=FALSE
);
ALTER TABLE potencias OWNER TO postgres;
```

Bibliografía

[1]. ACEVEDO M. E., YAÑEZ C. y FELIPE F., “Modelos asociativos”. *Komputer Sapiens*, Año 1, No. 2, Octubre 2008, pp 20-26.
[2]. YAÑEZ, C. “Memorias Asociativas αβ”. Tesis Doctoral. Instituto Politecnico Nacional, Centro de Investigación en Computación. (2002).
[3]. BATKE, J,M.E.G.W.P.S.T.: ”A query by humming system using mpeg-7 descriptors”. In: Audio Engineering Society Convention 116 (5 2004).
[4]. SHNEIDERMAN, B., PLAISANT, C., COHEN, M., JACOBS, S.: “Designing the User Interface: Strategies for Effective Human-Computer Interaction”. Addison Wesley.
[5]. LIN, C.C., CHEN, S.H., TRUONG, T.K., CHANG, Y.: Audio classification and categorization based on wavelets and support vector machine. IEEE Transactions on Speech and Audio Processing **13**(5) (2005) 644-651.
[6]. DORVERIC, V., RELJIN, N., RELJIN, I.: Identifying and retrieving of audio sequences by using wavelet descriptors and neural network with user’s assistance. EURO-CON2005 **05ex1255** (2005) 167-170.
[7]. WALKER, J., A primer on wavelets and their scientific applications, 2nd edition. Chapman Hall/CRC, Taylor and Francis Group (2008).
[8]. Ceballos, Fco. *Java: 2 Interfaces Gráficas y Aplicaciones para Internet*, Madrid, Alfaomega, 2006, p 241.
[9] Weka, revisado en Noviembre de 2010,
http://www/cs.waikato.ac.nz/ml/weka/index_downloading.html
[10] Formato de un archivo WAV canónico revisado en Noviembre de 2010.
https://ccrma.stanford.edu/courses/422/projects/WaveFormat/
[11] PostgreSQL, revisado en Noviembre de 2010. *http://www.postgresql.org/download/*
[12] PostgreSQL Driver JDBC, revisado en Noviembre de 2010, *http://jdbc.postgresql.org/*
[13] HERRERA, ALCANTARA, O.: Reconocimiento de archivos de sonido con redes neuronales y wavelets. Avances en Sistemas Inteligentes en México. Editores Miguel González y Oscar Herrera. Sociedad Mexicana de Inteligencia Artificial. ISBN 9786079536725.Primera edición, (2010)

Recursos

Computadora personal *Toshiba* L305D-S5895, que cuenta con procesador AMD *Turion*. 64 X2 a 2.0 GHz, 3 GB de memoria RAM.
Sistema Operativo *Ubuntu Desktop* 9.04 AMD64 *Jaunty Jackalope*.
Entorno de Desarrollo Integrado *Eclipse*.
Micrófono para PC.