**CHAPTER** **13:** **BOT** **PROGRAMMInG** **AnD** **nEURAL** **nETWORKS**

* Creación de un bot simple
* Análisis de texto
* Entrenamiento de un bot neuronal
* Uso de un bot neuronal

Los bots son programas informáticos que están diseñados para usar Internet de la misma manera que los humanos lo usan. Las redes neuronales pueden ser útiles en el desarrollo de bots. En este capítulo verá cómo se puede usar una red neuronal para ayudar a un bot a encontrar la información deseada en Internet.

Una discusión que explore la creación de bots podría llenar fácilmente un libro. Si desea obtener más información sobre la programación de bots, puede encontrar útil el libro HTTP Programming Rec- ipes for Java Bots (ISBN: 0977320669). Este libro presenta muchos algoritmos utilizadosparacrear bots. Los bots creados en este capítulo utilizarán parte del código de HTTP Programming Recipes for Java Bots.

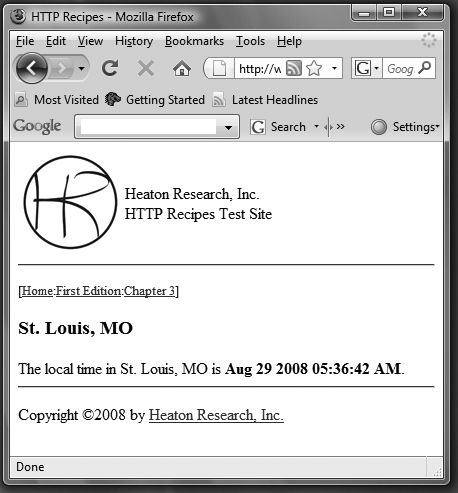
# Un bot simple

Un bot se utiliza generalmente para recuperar información de un sitio web. Antes de crear un bot complejo que haga uso de redes neuronales, será útil ver cómo se crea un bot simple. El bot simple obtendrá la hora actual para la ciudad de St. Louis, Missouri de un sitio web. El sitio que tiene los datos que estamos buscando se encuentra en la siguiente URL:

[**http://www.httprecipes.com/1/3/time.php**](http://www.httprecipes.com/1/3/time.php)

Si apunta un explorador a la dirección URL anterior, se mostrará la página que se muestra en la figura 13.1.

### Figura 13.1: Hora local en St. Louis, MO.



El código fuente del bot simple se muestra en el Listado 13.1.

### Listado 13.1: Un bot simple (SimpleBot.java)

paquete com.heatonresearch.book.introneuralnet.ch13;

importar java.io.\*; importar java.net.\*;

clase pública SimpleBot

{

/\*\*

\* El tamaño del búfer de descarga.

\*/

public static int BUFFER\_SIZE = 8192;

/\*\*

* Este método descarga el URL especificado en un Java
* cuerda. Este es un método muy simple, que se puede
* reutilizado en cualquier momento que necesite para obtener rápidamente todos los datos de
* una dirección URL específica.

\*

* @param url La URL que se va a descargar.
* @return El contenido de la dirección URL que se descargó.
* @produce IOException Thrown si se produce cualquier tipo de error.

\*/

public String downloadPage(URL url) lanza IOException

{

StringBuilder resultado = new StringBuilder(); byte buffer[] = nuevo byte[BUFFER\_SIZE];

InputStream s = url.openStream(); int tamaño = 0;

hacer

{

tamaño = s.read(búfer); if (tamaño != -1)

result.append(new String(buffer, 0, size));

} while (tamaño ! = -1);

return result.toString();

}

/\*\*

* Ejecute el ejemplo.

\*

* @param page La página a descargar.

\*/

public void go( Páginade cadena)

{

probar

{

URL u = nueva URL(página);

String str = downloadPage(u); System.out.println(str);

} catch (MalformedURLException e)

{

e.printStackTrace();

} catch (IOException e)

{

e.printStackTrace();

}

}

/\*\*

* Método principal típico de Java, cree un objeto y, a continuación,
* llame al método go de ese objeto.
* @param args Sitio web para acceder.

\*/

public static void main(String args[])

{

Módulo SimpleBot = nuevo SimpleBot(); Página de cadena;

if (args.length == 0)

página =  ["recipes.com/1/3/time.php http://www.http";](http://www.httprecipes.com/1/3/time.php)

más

página = args[0]; module.go(página);

}

}

El bot comienza creando un nuevo objeto **URL** para la página web deseada.

URL u = nueva  [URL("http://www.httprecipes.com/1/3/time.php");](http://www.httprecipes.com/1/3/time.php)

A continuación, el contenido de la página se descarga en un objeto **String.** El

**El** método downloadPage realiza esta operación.

String str = downloadPage(u);

Por último, el tiempo se extrae del código HTML. Esto se hace utilizando el método **de extracción.** Las etiquetas inicial y final que encierran los datos deseados se proporcionan al método **extract.** El tercer parámetro especifica la instancia de la etiqueta **<b>** que se va a encontrar. El número uno significa que estamos buscando la primera instancia del **<b>tag.**

System.out.println(extract(str, "<b>", "</b>", 1));

Los datos HTML de la página web de destino deben examinarse para ver qué etiquetas HTML encierran los datos deseados. El HTML ubicado en el sitio web que se muestra en la Figura 13.1 se muestra en el Listado 13.2.

### Listado 13.2: Datos HTML encontrados por el bot

<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<HTML>

<CABNSITA>

<>HTTP Recetas</TITLE>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<meta http-equiv="Cache-Control" content="no-cache">

<meta http-equiv="Content-Style-Type" content="text/css">

<link rel="alternate" type="application/rss+xml" title="RSS"  [href="http://www.httprecipes.com/1/12/rss2.xml">](http://www.httprecipes.com/1/12/rss2.xml)

</CABEZA>

<CUERBA>

<table border="0"><tr><td>

<a [href="http://www.httprecipes.com/"><img](http://www.httprecipes.com/)  src="/images/logo.gif" alt="Heaton Research Logo" border="0" width="100" height="100"></ a>

</td><td valign="middle">Heaton Research, Inc.<br> HTTP Recipes Test Site

</td></tr>

</tabla>

<hr><p><small>[<a href="/">Home</a>:<a href="/1/">First Edition</ a>:

<a href="/1/3/">Capítulo 3</a>]</small></p>

<h3>St. Louis, MO</h3>

La hora local en St. Loues, MO es <b>Jun 09 2008 10:27:46 PM</b>.

<hr>

<p>Copyright &copy;2008 por <a href="http://www.heatonresearch. com/">

Heaton Research, Inc.</a></p>

</CUERPO>

</HTML>

Como puede ver en la lista anterior, los datos que busca el bot se encuentran entre un **<b>tag** final y un **</b>tag** final. El bot hizo uso de dos métodos para descargar y luego extraer los datos. Estos dos métodos se tratarán en las dos secciones siguientes.

**Descarga de la** **página**

El contenido de una página web se descarga en un objeto **String** mediante el método **downloadPage.** La firma del método **downloadPage** se muestra aquí:

public String downloadPage(URL url) lanza IOException

En primer lugar, se crea un nuevo objeto **StringBuilder** para contener el contenido de la página web una vez descargada. Además, se crea un **búfer** para contener los datos a medida que se leen.

StringBuilder resultado = new StringBuilder(); byte buffer[] = nuevo byte[BUFFER\_SIZE];

A continuación, se crea un objeto **InputStream** para leer los datos del objeto **URL.**

InputStream s = url.openStream(); int tamaño = 0;

A continuación, se leen los datos hasta que se alcanza el final de la secuencia.

hacer {

tamaño = s.read(búfer);

Cuando se alcanza el final de la secuencia, el método **read** devolverá -1. Si no hemos llegado al final de la secuencia, el **búfer** se anexa al **resultado.**

si (tamaño ! = -1)

result.append(new String(buffer, 0, size));

} while (tamaño ! = -1);

Por último, el **resultado** se devuelve como un objeto **String.**

return result.toString();

El método **downloadPage** es muy útil para obtener una página web como un único objeto **String.** Una vez que se ha descargado la página, se utiliza el método **extract** para extraer datos de la página. El método **del extracto** se cubre en el ion siguiente de la secta.

## Extracción de los datos

Las páginas Web se pueden considerar como un objeto **String** grande. El proceso de dividir la cadena en datos utilizables se denomina análisis de cadenas. La firma para el método **de extracción** se muestra a continuación. El primer parámetro contiene la cadena que se va a analizar; los parámetros sec- ond y third especifican los tokens de principio y fin que se buscan. Por último, el cuarto parámetro especifica qué instancia del primer token está buscando.

public String extract(String str, String token1, String token2, int count)

Se crean dos variables para contener las ubicaciones de los dos tokens. Ambos están iniciadosa cero.

int location1, location2; ubicación1 = ubicación2 = 0;

A continuación, se encuentra la ubicación del primer token.

hacer

{

location1 = str.indexOf(token1, location1);

Si no se puede encontrar el primer token, se devuelve **null** para indicar que no hemos podido encontrar nada.

if (location1 == -1) devuelve null;

Si se encuentra el primer token, la variable **count** se reduce y continuamos buscando mientras haya más instancias que encontrar.

contar--;

} while (count > 0);

A continuación, intentamos localizar el segundo token. Si se encuentra el segundo token, su índice se almacena en la variable **location2.** Si el método **extract** no encuentra el segundo token, se devuelve null para indicar un error.

location2 = str.indexOf(token2, location1 + 1); if (location2 == -1)

devuelve null;

Por último, se devuelven los caracteres ubicados entre los dos tokens.

return str.substring(location1 + token1.length(), location2);

Aunque este bot es muy simple, demuestra los principios de la programación de bots. Las secciones restantes de este capítulo mostrarán cómo crear un bot basado en redes neuronales mucho más complejo.

# Introducción al bot neuronal

El bot basado en redes neuronales que se detalla a continuación se proporciona con el nombre de una persona famosa. Utiliza esta información para realizar una búsqueda en Yahoo y obtener información sobre la persona. El bot "lee" toda la información encontrada sobre la persona e intenta determinar el año de nacimiento correcto del individuo.

Hay tres modos distintos en los que se ejecuta este programa. Estos modos se resumen en el Cuadro 13.1.

### Tabla 13.1: Modos de bot neuronal cuando nacen

|  |  |
| --- | --- |
| **modo** | **propósito** |
| Gather | En este modo, el bot reúne artículos sobre personas famosas. Realiza una búsqueda de Yahoo para obtener un número de ar- ticles en cada uno. Dependiendo de su velocidad de Internet, este proceso puede tardar de quince minutos a varias horas. |
| Train | Utilizando los datos recopilados en el primer modo, se construye y entrena una red neuronal. En el nivelde error aceptable se alcanza, esta red neuronal se guarda. |
| Born | Una vez que se ha entrenado la red neuronal, se pueden ingresar nuevas personas famosas. A continuación, el bot intentará descubrir su año de nacimiento. |

El programa debe ejecutarse en el modo de **recopilación** antes de que se pueda entrenar. Asimismo, el programa debe ser entrenado antes de que pueda realizar la operación **Born.**

Hay varias constantes de configuración que se pueden modificar para este bot. Estos artículos se muestran en el listado 13.3.

### Listado 13.3: Configuración del bot neuronal (Config.java)

paquete com.heatonresearch.book.introneuralnet.ch13;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* Config: contiene los datos de configuración del bot.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

public class Config {

public final static int INPUT\_SIZE = 10; public final static int OUTPUT\_SIZE = 1; public final static int NEURONS\_HIDDEN\_1 = 20; public final static int NEURONS\_HIDDEN\_2 = 0;

doble estático final público ACCEPTABLE\_ERROR = 0,01; public final static int MINIMUM\_WORDS\_PRESENT = 3;

cadena final estática pública FILENAME\_GOOD\_TRAINING\_TEXT = "bornTrainingGood.txt";

cadena final estática pública FILENAME\_BAD\_TRAINING\_TEXT = "bornTrainingBad.txt";

public static final String FILENAME\_COMMON\_WORDS = "common.

csv";

neto";

hst";

}

public static final String FILENAME\_WHENBORN\_NET = "whenborn. public static final String FILENAME\_HISTOGRAM = "whenborn.

Las constantes de configuración se definen en la Tabla 13.2.

### Tabla 13.2: Configuración del bot neuronal

|  |  |  |
| --- | --- | --- |
| **constante** | **predeterminado** | **propósito** |
| INPUT\_SIZE | 10 | Número de neuronas de entrada. |
| OUTPUT\_SIZE | 1 | Número de neuronas de salida. |
| NEURONS\_HIDDEN\_1 | 20 | Número de neuronas en la primera capa oculta. |
| NEURONS\_HIDDEN\_2 | 0 | Número de neuronas en la segunda capa hid- den. |
| ACCEPTABLE\_ERROR | 0.01 | El nivel máximo de error aceptable. |
| MINIMUM\_WORDS\_PRESENT | 3 | Número mínimo de palabras para un sen- tence. |
| FILENAME\_GOOD\_TRAINING\_TEXT | bornTrainingGood.txt | Buenas frases reunidas. |
| FILENAME\_BAD\_TRAINING\_TEXT | bornTrainingBad.txt | Se acumularon malas sentencias. |
| FILENAME\_COMMON\_WORDS | común.csv | Palabras comunes en inglés. |
| FILENAME\_WHENBORN\_NET | whenborn.net | La red neuronal entrenada guardada. |
| FILENAME\_HISTOGRAM | whenborn.hst | El histoguardado- gramo de palabras comunes. |

Ahora examinaremos cada modo de operación para el bot neuronal.

# Recopilación de datos de entrenamiento para el bot neuronal

El bot basado en neuronales funciona realizando una búsqueda de Yahoo en una persona de interés. Todos los artículos devueltos por Yahoo se descomponen en oraciones y se escanean para el año de nacimiento correcto. Una lista que contiene a personas famosas y sus años de nacimiento se utiliza como datos de entrenamiento. Esta lista se almacena en un archivo llamado "famous.csv". En el listado 13.4 se muestra una muestra de los datos.

### Listado 13.4: Personajes Famosos

Persona,Año

Abdullah bin Abdul Aziz Al Saud,1924 Al Gore,1948

Alber Elbaz,1961 América Ferrera,1984 Amr Khaled,1967 Angela Merkel,1954 Anna Netrebko,1971

Arnold Schwarzenegger,1947 Ayatullah Ali Khamenei,1939 Barack Obama,1961

Bernard Arnault,1949

Bill Gates,1955

Alfilerillo Pitt,1963

...

Warren Buffett,1930 Wesley Autrey,1956

Guillermo Jefferson Clinton,1946

Youssou N'Dour,1959

Zeng Jinyan, 1983

Con esta lista, se realizará una búsqueda en Yahoo para recopilar información de cada una de estas personas.

**Recopilación de** **datos**

Para recopilar datos para el entrenamiento, se ejecuta la clase **GatherForTrain.** La clase **GatherForTrain** comienza a realizar búsquedas en Yahoo y a recopilar datos de todas las personas famosas. La clase **GatherForTrain** se compila para ser multiproceso. Esto acelera el procesamiento, incluso en un equipo con un solo procesador. Debido a que el programa a menudo debe esperar a que los muchos sitios web que visita respondan, mejora la eficiencia para acceder a varios sitios web a la vez. El código fuente de la clase **GatherForTrain** se muestra en el listado 13.5.

### Listado 13.5: Recopilación de datos de entrenamiento (GatherForTrain.java)

paquete com.heatonresearch.book.introneuralnet.ch13.gather; importar java.io.FileOutputStream;

importar java.io.IOException; importar java.io.OutputStream; importar java.io.PrintStream; importar java.util.ArrayList; importar java.util.Collection; importar java.util.Date; importar java.util.Vector;

importar java.util.concurrent. Invocable;

importar java.util.concurrent. ExecutorService; importar java.util.concurrent.Executors;

importar com.heatonresearch.book.introneuralnet.ch13.Config;

importar com.heatonresearch.book.introneuralnet.ch13.ScanReportable; importar com.heatonresearch.book.introneuralnet.common.ReadCSV;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* GatherForTrain: Recopilar datos de entrenamiento para la red neuronal.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

public class GatherForTrain implements ScanReportable { public static final boolean LOG = true;

public static final int THREAD\_POOL\_SIZE = 20;

/\*\*

* + El método main procesa los argumentos de la línea de comandos
  + y luego llama
  + método de proceso para determinar el año de nacimiento.

\*

* + @param args
  + Contiene la clave de Yahoo y el sitio para buscar.

\*/

public static void main(final String args[]) { try {

final GatherForTrain cuando = nuevo GatherForTrain(); when.process();

} catch (excepción final e) { e.printStackTrace();

}

}

grupo executorservice privado;

private final Collection<String> trainingDataGood = new Vector<String>();

private final Collection<String> trainingDataBad = new Vector<String>();

private int currentTask; private int totalTasks;

/\*\*

* + Se llama a este método para determinar el año de nacimiento de un
  + persona. Se
  + obtiene 100 páginas web que Yahoo devuelve para esa persona.
  + Cada uno de ellos
  + a continuación, se busca en las páginas el año de nacimiento de esa persona.
  + Que cada año
  + se selecciona el mayor número de veces se selecciona como
  + el año de nacimiento.

\*

* + @ nombre param
  + El nombre de la persona que está viendo el
  + año de nacimiento para.
  + @lanza IOException
  + Se produce si se produce un error de comunicación.

\*/

public void process() produce IOException { this.pool = Ejecutores

. newFixedThreadPool( GatherForTrain.THREAD\_POOL\_SIZE);

final Collection<Callable<Integer>> tasks = new ArrayList<Callable<Integer>>();

final ReadCSV famoso = nuevo ReadCSV("famoso.csv"); informe(

"Construyendo datos de entrenamiento a partir de la lista de personas famosas."); fecha final de inicio = nueva fecha();

while (famous.next()) {

final String name = famous.get("Persona"); final int year = famous.getInt("Año");

}

probar {

trabajador final de CollectionWorker =

nuevo CollectionWorker(esto, nombre, año);

tasks.add(trabajador);

this.totalTasks = tasks.size(); this.currentTask = 1; this.pool.invokeAll(tareas); this.pool.shutdownNow();

informe("Hecho la recopilación de datos de Internet.");

} catch (final InterruptedException e) {

TODO Bloque catch generado automáticamente e.printStackTrace();

}

longitud larga = (nueva Fecha()). getTime() - started.getTime();

longitud /= 1000L; longitud /= 60;

System.out.println("Tomó " + longitud

+ " minutos para recopilar datos de entrenamiento de Internet."); System.out.println("Escribir archivo de entrenamiento"); writeTrainingFile();

}

public void receiveBadSentence(oración de cadena final) { this.trainingDataBad.add(oración);

}

public void receiveGoodSentence(oración de cadena final) { this.trainingDataGood.add(oración);

}

public void report(final String str) { synchronized (this) {

System.out.println(str);

}

}

public void reportDone(cadena decadena final) { report(this.currentTask + "/" + this.totalTasks

+ ":" + cadena); this.currentTpreguntar++;

}

private void writeTrainingFile() throws IOException { OutputStream fs = new FileOutputStream(

Config.FILENAME\_GOOD\_TRAINING\_TEXT);

PrintStream ps = nuevo PrintStream(fs);

for (final String str : this.trainingDataGood) { ps.println(str.trim());

}

ps.close();

fs.close();

fs = new FileOutputStream( Config.FILENAME\_BAD\_TRAINING\_TEXT);

ps = new PrintStream(fs);

for (final String str : this.trainingDataBad) { ps.println(str.trim());

}

ps.close();

fs.close();

}

}

El método de **proceso** realiza casi todo el trabajo en el **GatherForTrain**

clase. La firma para el método **de proceso** se muestra aquí:

public void process() produce IOException

En primer lugar, se crea un nuevo grupo de subprocesos.

this.pool = Ejecutores

. newFixedThreadPool(GatherForTrain.THREAD\_POOL\_SIZE); final Collection<Callable<Integer>> tasks = new ArrayList<Callable<Integer>>();

A continuación, la clase **ReadCSV** se utiliza para leer la lista de valores separados por comas de las personas fa- mous. Los archivos separados por comas se introdujeron en el capítulo 10.

final ReadCSV famoso = nuevo ReadCSV("famoso.csv"); informe("Construyendo datos de entrenamiento a partir de la lista de personas famosas."); fecha final iniciado = new Date();

Si bien todavía hay datos por leer, seguimos recorriendo la lista de personas famosas.

while (famous.next()) {

La persona famosa y su año de nacimiento se obtienen del archivo CSV.

final String name = famous.get("Persona"); final int year = famous.getInt("Año");

Se crea un objeto **CollectionWorker** para procesar realmente a esta persona y se agrega al grupo de subprocesos.

trabajador final de CollectionWorker = nuevo CollectionWorker(esto, nombre, año);

tasks.add(trabajador);

}

A continuación, se ejecuta el grupo de subprocesos.

probar {

this.totalTasks = tasks.size(); this.currentTask = 1; this.pool.invokeAll(tareas); this.pool.shutdownNow();

informe("Hecho la recopilación de datos de Internet.");

} catch (final InterruptedException e) { e.printStackTrace();

}

Una vez que el grupo de subprocesos termina, el procesamiento se completa y la cantidad de

se muestra el tiempo que tardó el procesamiento.

longitud larga = (nueva Fecha()). getTime() - started.getTime(); longitud /= 1000L;

longitud /= 60; System.out.println("Tomó " + longitud

+ " minutos para recopilar datos de entrenamiento de Internet."); System.out.println("Escribir archivo de entrenamiento"); writeTrainingFile();

Por último, los datos se escriben en el archivo de entrenamiento.

## Compatibilidad con multithreading

El **collectionworker** proporciona el subprocesamiento múltiple para el modo de recopilación

la clase **CollectionWorker** se muestra en el listado 13.6.

### Listado 13.6: Trabajador de colección (CollectionWorker.java)

paquete com.heatonresearch.book.introneuralnet.ch13.gather;

importar java.io.IOException; importar java.net.URL;

importar java.util. Colección;

importar java.util.concurrent. Invocable;

importar com.heatonresearch.book.introneuralnet.ch13.Text;

importar com.heatonresearch.book.introneuralnet.common.YahooSearch;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* CollectionWorker: clase de trabajo en un grupo de subprocesos utilizado para
* recopilar frases de la web.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

public class CollectionWorker implementa Callable<Integer> { private final GatherForTrain bot;

/\*

* + Objeto de búsqueda que se va a utilizar.

\*/

búsqueda final privada de YahooSearch;

nombre de cadena final privado; año privado final int;

public CollectionWorker(bot GatherForTrain final, nombre de cadena final,

final int year) { this.bot = bot; this.name = nombre; this.year = año;

this.search = nuevo YahooSearch();

}

public Integer call() produce Exception { try {

scanPerson(this.name, this.year); this.bot.reportDone(this.name + ", hecho el escaneo.");

} catch (excepción final e) { this.bot.reportDone(this.name + ", error encontrado."); e.printStackTrace();

lanzar e;

}

devuelve null;

}

private void scanPerson(final String name, final int year) throws IOException {

final Collection<URL> c = this.search.search(nombre); int i = 0;

for (URL final u : c) { try {

i++;

Text.checkURL(this.bot, u, año);

} catch (final IOException e) {

}

}

}

}

El grupo de subprocesos inicia el programa de trabajo llamando al método de **llamada.** Como puedas

ver en la lista del método de **llamada** anterior, hace poco más que llamar a la

**método** scanPerson. La firma del método **scanPerson** se muestra aquí:

private void scanPerson(nombre de cadena final, último año int) produce IOException

En primer lugar, se obtiene una colección de DIRECCIONES URL de Yahoo para el famoso per- hijo especificado.

final Collection<URL> c = this.search.search(nombre); int i = 0;

Para cada **dirección URL** que se obtiene, se llama al método **checkURL** de la clase Text.

La clase **Text** se tratará en la siguiente sección.

for (URL final u : c) { try {

i++;

Text.checkURL(this.bot, u, año);

} catch (final IOException e) {

}

}

Si se encuentra un error, es probable que se haya obtenido una URL no válida de Yahoo.

No se trata de un hecho inusual; tales URL simplemente se ignoran.

## Análisis de sitios web

Como ha visto en la última sección, los sitios web se analizan mediante la clase denominada **Text.**

La clase **Text** se muestra en el listado 13.7.

### Listado 13.7: Análisis de sitios web (Texto.java)

paquete com.heatonresearch.book.introneuralnet.ch13;

importar java.io.IOException; importar java.io.InputStream; importar java.net.URL;

importar java.net.URLConnection; importar java.util.StringTokenizer;

importar com.heatonresearch.httprecipes.html.ParseHTML;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* Texto: Utilidades de análisis de texto.

\*

* @versión 2.1

\*/

public class Texto {

/\*\*

* + Compruebe la dirección URL especificada para un año de nacimiento. Esto
  + Se producen si se encuentra una frase que tiene la palabra
  + born y un valor numérico menor que 3000.

\*

* + @param url
  + Dirección URL que se va a comprobar.
  + @lanza IOException
  + Se produce si se produce un error de comunicación.

\*/

public static void checkURL(informefinal de ScanReportable, url final url,

final Integer desiredYear) produce IOException { int ch;

oración final de StringBuilder = new StringBuilder(); Cadena ignoreUntil = null;

urlconnection final http = url.openConnection(); http.setConnectTimeout(1000); http.setReadTimeout(1000);

InputStream final es = http.getInputStream(); final ParseHTML html = new ParseHTML(is);

hacer {

ch = html.read();

si ((ch ! = -1) && (ch != 0)

&&(ignoreUntil == null)) { if (".?!". indexOf(ch) != -1) {

cadena final str =

oración.toString(); año int final =

Text.extractYear(str);

if (desiredYear == null) {

buscando cualquier año si (año ! = -1) {

informe.

receiveGoodSentence(str);

}

} else {

buscando un año específico si (año == desiredYear) {

informe.

receiveGoodSentence(str);

} else if (año ! = -1) {

informe.

receiveBadSentence(str);

}

}

sentence.setLength(0);

} else if (ch == ' ') {

if ((oración.longitud() > 0)

&& (sentence. charAt(sentence.length() - 1) != ' ')) {

sentence.append(' ');

}

} else if ((ch ! = '\n') && (ch != '\t') && (ch != '\r')) {

if ((ch) < 128) {

frase.anexar((char) ch);

}

}

} else if (ch == 0) {

borrar cualquier cosa antes de una etiqueta de cuerpo si (html.getTag(). getName().

equalsIgnoreCase("cuerpo")

|| html.getTag(). getName(). equalsIgnoreCase("br")

|| html.getTag(). getName(). equalsIgnoreCase("li")

|| html.getTag(). getName(). equalsIgnoreCase("p")

|| html.getTag(). getName(). equalsIgnoreCase("h1")

|| html.getTag(). getName(). equalsIgnoreCase("h2")

|| html.getTag(). getName(). equalsIgnoreCase("h3")

|| html.getTag(). getName(). equalsIgnoreCase("td")

|| html.getTag(). getName(). equalsIgnoreCase("th")) {

sentence.setLength(0);

etiquetas de estilo}

ignorar todo entre el script y

if (ignoreUntil == null) {

if (html.getTag(). getName(). equalsIgnoreCase("script")) {

ignoreUntil = "/script";

} else if (html.getTag(). getName()

. equalsIgnoreCase("estilo")) {

ignoreUntil = "/estilo";

}

} else {

if (html.getTag(). getName()

. equalsIgnoreCase(ignoreUntil)) {

ignoreUntil = null;

}

}

agregue un espacio después de la etiqueta if (sentence.length() > 0) {

if (sentence.charAt( sentence.length() - 1) != ' ') {

sentence.append(' ');

}

}

}

} mientras que (ch ! = -1);

}

/\*\*

* + Examinar una oración y ver si contiene la palabra
  + nacido y un número.

\*

* + @param oración
  + La frase a buscar.
  + @return Es el número que se encontró.

\*/

public static int extractYear(final String sentence) { int result = -1;

tok final de StringTokenizer =

nuevo StringTokenizer(oración); while (tok.hasMoreTokens()) {

palabra de cadena final = tok.nextToken();

probar {

resultado = Integer.parseInt(palabra);

If ((resultado < 1600) || (resultado > 2100)) { resultado = -1;

}

} catch (final NumberFormatException e) {

}

}

resultado devuelto;

}

}

Se llama al método **checkURL** de la clase **Text** para examinar una página Web y buscar

para sentencias. Estas oraciones se dividen en dos categorías: buenas y malas. Si una sentencia contiene un año, y es el año del nacimiento de la persona famosa, entonces esta sentencia es una buena frase; de lo contrario, es una mala sentencia. Las dos listas creadas como resultado de esta clasificación se utilizan para entrenar la red neural. La firma para el método **checkURL** se muestra aquí:

public static void checkURL(informe final de ScanReportable, url final url,

final Integer desiredYear) produce IOException

Como puede ver, se pasan tres parámetros a este método. El primero, denominado **report**, especifica el objeto que recibirá las listas de frases buenas y malas que generaráeste método. El segundo, denominado **url**,especifica la dirección **URL** del sitio web que se va a analizar. El tercero, denominado **desiredYear**, especifica el año enque el persen nació realmente .

En primer lugar, la variable **ch** se declara para contener el carácter actual. La **oración** variable se declara para contener la oración actual, e **ignoreUntil** se declara para contener una etiqueta HTML en la que nos basaremos para indicar cuándo debemos comenzar a procesarlos datos denuevo. Por ejemplo, si encontramos la etiqueta HTML **<script>,**estableceremos **ignoreUntil** en **</script>y** no procesaremos ninguno de los códigos entre **<script>y** el

</script>tag. Los datos entre estas etiquetas son Javascript y no serán útiles para determinar un año de nacimiento.

int ch;

oración final de StringBuilder = new StringBuilder(); Cadena ignoreUntil = null;

Se abre una conexión HTTP a la dirección URL especificada. Se especifican tiempos de espera de1.000 m illisec- onds, o un segundo. Procesaremos un gran número de páginas y no queremos esperar demasiado tiempo para que una devuelva un error. Si los datos no están listos en un segundo, pasamos a la página siguiente.

urlconnection final http = url.openConnection(); http.setConnectTimeout(1000); http.setReadTimeout(1000);

InputStream final es = http.getInputStream(); final ParseHTML html = new ParseHTML(is);

Con la conexión establecida, comenzamos a leer desde la página HTML. Este método utiliza el analizador HTML proporcionado en el libro HTTP Programming Recipes for Java Bots (ISBN: 0977320669).

hacer {

ch = html.read();

Si el valor de **ch** es -1, se ha encontrado una etiqueta HTML. De lo contrario, **ch** será el siguiente carácter del texto. Si el carácter es un punto, signo de interrogación o signo de ex-clamación, entonces sabemos que hemos llegado al final de una oración, a menos que estemos ignorando los datos hasta que lleguemos a una etiqueta, en cuyo caso **ignoreUntil** no será **null.**

si ((ch ! = -1) && (ch != 0) && (ignoreUntil == null)) { if (".?!". indexOf(ch) != -1) {

Cuando hemos acumulado una oración completa, convertimos la **oración** **StringBuffer** en una **String** y determinamos si la oración contiene o no un año. Para ello se utiliza la clase **Text.** La clase **Text** no hace nada más que utilizar **StringTokenizer** para dividir la oración en cada espacio en blanco y buscar durante un año.

final String str = sentence.toString(); final int year = Text.extractYear(str);

A continuación, determinamos si se ha especificado un año en particular.

if (desiredYear == null) {

Si no, entonces se informa el año encontrado y se le dice al usuario que esto es una "buena frase". Si no se encontró ningún año, la variable **año** es -1.

si (año ! = -1) { report.receiveGoodSentence(str);

}

} else {

Si estamos buscando un año específico, entonces determinamos si el año encontrado coincide con la variable **desiredYear** y lo reportamos como una oración "buena" o "mala". Si no se encontró ningún año, entonces no informamos nada.

if (year == desiredYear) { report.receiveGoodSentence(str);

} else if (año ! = -1) { report.receiveBadSentence(str);

}

}

La **oración** **StringBuffer** se borra para dar paso a la siguiente sen- tence.

sentence.setLength(0);

Si se producen varios espacios juntos, se recortan a un solo espacio.

} else if (ch == ' ') {

if ((oración.longitud() > 0)

&& (sentence.charAt(sentence.length() - 1) != ' '))

{

sentence.append(' ');

}

Los siguientes caracteres de formato y los caracteres por encima de 128 no se han pro- cessed.

} else if ((ch ! = '\n') && (ch != '\t') && (ch != '\r')) { if ((ch) < 128) {

frase.anexar((char) ch);

}

}

Las siguientes etiquetas comienzan nuevas secciones. La **frase** se borra si uno es encoun- tered y empezamos de nuevo.

} else if (ch == 0) {

borrar cualquier cosa antes de una etiqueta de cuerpo

if (html.getTag(). getName(). equalsIgnoreCase("cuerpo")

|| html.getTag(). getName(). equalsIgnoreCase("br")

|| html.getTag(). getName(). equalsIgnoreCase("li")

|| html.getTag(). getName(). equalsIgnoreCase("p")

|| html.getTag(). getName(). equalsIgnoreCase("h1")

|| html.getTag(). getName(). equalsIgnoreCase("h2")

|| html.getTag(). getName(). equalsIgnoreCase("h3")

|| html.getTag(). getName(). equalsIgnoreCase("td")

|| html.getTag(). getName(). equalsIgnoreCase("th")) { sentence.setLength(0);

}

Si aún no estamos en el modo "ignorar hasta", entonces determinamos si ahora debemos cambiar a este modo.

ignore todo lo que hay entre las etiquetas de script y style if (ignoreUntil == null) {

Si la etiqueta encontrada es una etiqueta **<script>,** ignoramos todo el texto hasta que finalice

**se encuentra </script>tag.**

if (html.getTag(). getName(). equalsIgnoreCase("script")) { ignoreUntil = "/script";

} else if (html.getTag(). getName()

. equalsIgnoreCase("estilo")) { ignoreUntil = "/style";}

} else {

Si nos encontramos con la etiqueta que hemos estado buscando, entonces borramos el

**ignoreUntil** variable y comience a procesar caracteres de nuevo.

if (html.getTag(). getName(). equalsIgnoreCase(ignoreUntil))

{

ignoreUntil = null;

}

}

Se agrega un espacio después de encontrar cualquier etiqueta.

agregue un espacio después de la etiqueta if (sentence.length() > 0) {

if (sentence.charAt(sentence.length() - 1) != ' ') { sentence.append(' ');

}

}

}

} mientras que (ch ! = -1);

El procesamiento continúa hasta que se alcanza el final. Al final de este proceso, el sen- tences se clasifican en buenos y malos. Sólo se consideran aquellas sentencias que contenían un año válido.

# Entrenamiento del bot neuronal

La clase **TrainBot** se ejecuta para entrenar el bot neuronal. Esta clase hace uso de varias otras clases en el paquete **train.** La clase **TrainBot** se muestra en List- ing 13.8.

### Listado 13.8: Entrenar al bot (TrainBot.java)

paquete com.heatonresearch.book.introneuralnet.ch13.train; importar java.io.IOException;

importar com.heatonresearch.book.introneuralnet.ch13.CommonWords; importar com.heatonresearch.book.introneuralnet.ch13.Config; importar com.heatonresearch.book.introneuralnet.ch13.NetworkUtil;

import com.heatonresearch.book.introneuralnet.neural.feedforward.

FeedforwardNetwork;

importar com.heatonresearch.book.introneuralnet.neural.feedforward.

tren. Tren;

importar com.heatonresearch.book.introneuralnet.neural.feedforward.

train.backpropagation. Retropropagación;

importar com.heatonresearch.book.introneuralnet.neural.util.

SerializeObject;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* TrainBot:Entrene el bot con los datos recopilados.

\*/

clase pública TrainBot {

public static void main(final String args[]) { try {

final TrainBot trainBot = nuevo TrainBot(); trainBot.process();

} catch (excepción final e) { e.printStackTrace();

}

}

private int sampleCount; commonwords privados comunes; entrada doble privada[ ][]; ideal doble privado[][];

red privada FeedforwardNetwork; privado AnalyzeSentences goodAnalysis; privado AnalyzeSentences badAnalysis; WordHistogram histogramGood; Histograma de WordHistogramBad ;

entrenamiento final privadoSet trainingSet; public TrainBot() lanza IOException {

this.com mon = new CommonWords(

Config.FILENAME\_COMMON\_WORDS);

this.trainingSet = nuevo TrainingSet();

}

private void allocateTrainingSets() { this.input = new double[this.sampleCount]

[Config.INPUT\_SIZE];

this.ideal = new double[this.sampleCount] [Config.OUTPUT\_SIZE];

}

private void copyTrainingSets() { int index = 0;

primero la entrada

for (final double[] array : this.trainingSet. getInput()) {

System.arraycopy(matriz, 0, this.input[índice],

0, array.length); index++;

}

índice = 0;

en segundo lugar el ideal

for (final double[] array : this.trainingSet. getIdeal()) {

System.arraycopy(matriz, 0, this.ideal[índice],

0, array.length); index++;

}

}

public void process() produce IOException { this.network = NetworkUtil.createNetwork();

System.out.println("Preparación de conjuntos de entrenamiento..."); this.common = new CommonWords(

Config.FILENAME\_COMMON\_WORDS);

this.histogramGood = nuevo WordHistogram(this.common); este.histogramBad = nuevo WordHistogram(this.common);

cargar las buenas palabras this.histogramGood.buildFromFile(

Config.FILENAME\_GOOD\_TRAINING\_TEXT);

this.histogramGood.buildComplete();

cargar las palabras incorrectas this.histogramBad.buildFromFile(

Config.FILENAME\_BAD\_TRAINING\_TEXT);

this.histogramBad.buildComplete();

eliminar palabras de puntuación baja this.histogramGood

. removeBelow((int) this.histogramGood.calculateMean());

this.histogramBad.removePercent(0,99);

eliminar palabras comunes this.histogramGood.removeCommon(this.histogramBad);

this.histogramGood.trim(Config.INPUT\_SIZE);

this.goodAnalysis = new AnalyzeSentences( this.histogramGood,

Config.INPUT\_SIZE); this.badAnalysis = new AnalyzeSentences(

this.histogramGood, Config.INPUT\_SIZE);

this.goodAnalysis.process(this.trainingSet, 0.9, Config.FILENAME\_GOOD\_TRAINING\_TEXT);

this.badAnalysis.process(this.trainingSet, 0.1, Config.FILENAME\_BAD\_TRAINING\_TEXT);

this.sampleCount = this.trainingSet.getIdeal().size();

System.out

. println("Procesando "

+ this.sampleCount + " conjuntos de entrenamiento."); allocateTrainingSets();

copyTrainingSets();

trainNetworkBackpropBackprop(); SerializeObject.save(Config.FILENAME\_WHENBORN\_NET,

this.network); SerializeObject.save(Config.FILENAME\_HISTOGRAM,

this.histogramGood); System.out.println("Entrenamiento completo.");

}

private void trainNetworkBackpropBackprop() { final Train train =

new Backpropagation(this.network, this.input, this.ideal, 0.7, 0.7);

int época = 1;

hacer {

train.iteration(); System.out.println("Backprop:Iteración #"

+ época + " Error:"

+ train.getError()); época++;

} while ((train.getError() > Config.ACCEPTABLE\_ERROR));

}

}

El método **process** es el primer método al que llama el método **main** de la clase **TrainBot.** El método **de proceso** se analizará en la siguiente sección.

**Procesamiento de los** **conjuntos** **de entrenamiento**

El método **de proceso** comienza creando una red neuronal e informando de que se están preparando los conjuntos de entrenamiento.

this.network = NetworkUtil.createNetwork(); System.out.println("Preparación de conjuntos de entrenamiento...");

Solo las 1.000 palabras en inglés más comunes pueden influir en la red neuronal. Se analizarán las ocurrencias de estas palabras comunes en los conjuntos de oraciones buenas y malas. Los histogramas, que se tratarán más adelante en este capítulo, realizan un seguimiento del número de apariciones de cada una de las palabras comunes en las oraciones buenas y malas.

this.common = new CommonWords(Config.FILENAME\_COMMON\_WORDS); this.histogramGood = nuevo WordHistogram(this.common); this.histogramBad = nuevo WordHistogram(this.common);

En primer lugar, las buenas palabras se cargan en su histograma.

this.histogramGood.buildFromFile(Config.FILENAME\_GOOD\_TRAINING\_ TEXT);

this.histogramGood.buildComplete();

A continuación, las palabras malas se cargan en su histograma.

this.histogramBad.buildFromFile(Config.FILENAME\_BAD\_TRAINING\_TEXT); this.histogramBad.buildComplete();

Nos gustaría eliminar cualquier palabra que aparece en las listas de entrenamiento bueno y malo. Los recuentos de apariciones se utilizan para recortar las listas; es probable que se considereque sepuede superponer. Para superar esta superposición, eliminamos cualquier palabra del histograma bueno que esté por debajo del número promedio de apariciones. Luego eliminamos el 99% más bajo de malas palabras.

this.histogramGood

. removeBelow((int) this.histogramGood.calculateMean()); this.histogramBad.removePercent(0,99);

A continuación, eliminamos las palabras que aparecen en los histogramas buenos y malos.

eliminar palabras comunes this.histogramGood.removeCommon(this.histogramBad);

Por último, recortamos el buen histogram al número de entradas.

this.histogramGood.trim(Config.INPUT\_SIZE);

Luego analizamos las frases buenas y malas. Esto nos permitirá crear una matriz de entrada y una matriz ideal basada en cada frase. La matriz de entrada refleja cuántas de las buenas palabras estaban presentes en cada oración. La matriz de salida re- flects si esta oración contiene el año de nacimiento o no. Comenzamos asignando dos objetos **AnalyzeSentence.**

this.goodAnalysis = new AnalyzeSentences(this.histogramGood, Config.INPUT\_SIZE);

this.badAnalysis = new AnalyzeSentences(this.histogramGood, Config.INPUT\_SIZE);

Luego procesamos tanto las oraciones buenas como las malas. Todos ellos se añadirán a la

**colección** trainingSet que se pasa al método.

this.goodAnalysis.process(this.trainingSet, 0.9, Config.FILENAME\_GOOD\_TRAINING\_TEXT);

this.badAnalysis.process(this.trainingSet, 0.1, Config.FILENAME\_BAD\_TRAINING\_TEXT);

Observe los valores de 0,9 y 0,1 anteriores. Estos son los valores de salida ideales. Cuanto más probable es que el valor de la neurona de salida sea de 0,9, más probable es que la oración contenga el año de nacimiento de la persona famosa. Cuanto más cerca esté de 0,1, menos probable será.

A continuación, informamos del número de conjuntos de entrenamiento que se recopilaron.

this.sampleCount = this.trainingSet.getIdeal().size();

System.out

. println("Processing " + this.sampleCount + " training sets.");

Para permitir que los conjuntos de entrenamiento crezcan fácilmente, se almacenan como una colección. Con el fin de utilizarlos para entrenar la red neuronal, deben convertirse en la matriz 2-dimen- sional habitual, al igual que las muestras utilizadas para entrenar las redes neuronales anteriores en este book. Comenzamos asignando los conjuntos de entrenamiento.

allocateTrainingSets();

A continuación, la colección **trainingSet** se copia en los conjuntos de entrenamiento reales.

copyTrainingSets();

La red neuronal ahora se entrena mediante la retropropagación.

trainNetworkBackpropBackprop();

A continuación, se guardan la red neuronal y el histograma.

SerializeObject.save(Config.FILENAME\_WHENBORN\_NET, this.network); SerializeObject.save(Config.FILENAME\_HISTOGRAM, this.histogramGood);

System.out.println("Entrenamiento completado." );

La capacitación ya está completa.

## Análisis de histogramas de oraciones

Un histograma es una progresión lineal de frecuencias. Los histogramas de esta aplicación- catión son administrados por la clase **WordHistogram.** Se utilizan para almacenar el fre- quency con el que cada palabra común aparece en una oración. La clase **WordHistogram** se muestra en el listado 13.9.

### Listado 13.9: Administración de histogramas (WordHistogram.java)

paquete com.heatonresearch.book.introneuralnet.ch13.train;

importar java.io.BufferedReader; importar java.io.FileInputStream; importar java.io.IOException; importar java.io.InputStream; importar java.io.InputStreamReader; importar java.io.Serializable; importar java.util.HashMap;

importar java.util. Mapa; importar java.util.Set;

importar java.util. StringTokenizer; importar java.util.TreeSet;

importar com.heatonresearch.book.introneuralnet.ch13.CommonWords; importar com.heatonresearch.book.introneuralnet.ch13.Config;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* WordHistogram: Construir un histograma de cuántas apariciones de
* cada palabra.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

public class WordHistogram implementa Serializable {

/\*\*

* + Identificador de serie para esta clase.

\*/

private static final long serialVersionUID = 4712929616016273693L;

public static void main(final String args[]) {

probar {

final CommonWords common = new CommonWords( Config.FILENAME\_COMMON\_WORDS);

final WordHistogram histogramGood = nuevo WordHistogram(común);

histograma final de WordHistogramBad = nuevo WordHistogram(común);

cargar las palabras buenas histogramGood.buildFromFile(

Config.FILENAME\_GOOD\_TRAINING\_TEXT);

histogramGood.buildComplete();

cargar las palabras incorrectas histogramBad.buildFromFile(

Config.FILENAME\_BAD\_TRAINING\_TEXT);

histogramBad.buildComplete();

eliminar palabras de puntuación baja histogramGood.removeBelow((int)

histogramGood.calculateMean()); histogramBad.removePercent(0,99);

eliminar palabras comunes histogramGood.removeCommon(histogramBad);

histogramBad.removeCommon(histogramGood);

System.out.println("Buenas palabras"); para (elemento histogramelemento final:

histogramGood.getSorted()) {

System.out

. println(

element.getWord() + ":" + element.getCount());

}

System.out.println("\n\n\n"); System.out.println("Palabras malas"); para (elemento histogramelemento final:

histogramBad.getSorted()) {

System.out

. println(element.getWord()

+ ":" + element.getCount());

}

} catch (excepción final e) { e.printStackTrace();

}

}

commonwords finales privados comunes;

private final Map<String, HistogramElement> histogram = new HashMap<String, HistogramElement>();

private final Set<HistogramElement> sorted = new TreeSet<HistogramElement>();

public WordHistogram(final CommonWords common) { this.common = common;

}

public void buildComplete() { this.sorted.clear();

this.sorted.addAll(this.histogram.values());

}

public void buildFromFile(final InputStream is) throws IOException {

Línea de cadena;

final BufferedReader br = nuevo

BufferedReader(nuevo InputStreamReader(is)); while ((line = br.readLine()) != null) {

buildFromLine(línea);

}

br.close();

}

public void buildFromFile(final String filename) throws IOException {

fis final de FileInputStream =

new FileInputStream(nombre de archivo); buildFromFile(fis);

fis.close();

}

public void buildFromLine(línea de cadena final) {

final StringTokenizer tok = nuevo StringTokenizer(línea); while (tok.hasMoreTokens()) {

palabra de cadena final = tok.nextToken(); buildFromWord(palabra);

}

}

public void buildFromWord(String word) { word = word.trim(). toLowerCase();

if (this.common.isCommonWord(word)) { HistogramElement element =

this.histogram.get(palabra); if (elemento == null) {

elemento = nuevo HistogramElement(palabra, 0); this.histogram.put(palabra, elemento);

}

element.increase();

}

}

public double calculateMean() { int total = 0;

for (elemento histogramelemento final: this.sorted) { total += element.getCount();

}

return (double) total / (double) this.sorted.size();

}

public double[] compact(final String line) {

final double[] result = new double[this.sorted.size()]; final StringTokenizer tok = nuevo StringTokenizer(línea);

while (tok.hasMoreTokens()) {

final Palabra de cadena = tok.nextToken(). toLowerCase();

final int rank = getRank(palabra); si (rango ! = -1) {

result[rango] = 0,9;

}

}

resultado devuelto;

}

public int count(final String line) { int result = 0;

final StringTokenizer tok = new StringTokenizer( línea);

while (tok.hasMoreTokens()) {

final Palabra de cadena = tok.nextToken(). toLowerCase();

final int rank = getRank(palabra); si (rango ! = -1) {

resultado++;

}

}

resultado devuelto;

}

public HistogramElement get(final Stringword) { return this.histogram.get(word.toLowerCase());

}

public CommonWords getCommon() { return this.common;

}

public int getRank(final String word) { int result = 0;

for (elemento histogramelemento final : this.sorted){ if (element.getWord(). equalsIgnoreCase(palabra)) {

resultado devuelto;

}

resultado++;

}

retorno -1;

}

public Set<HistogramElement> getSorted() { return this.sorted;

}

public void removeBelow(valor int final) {

final Object[] elements = this.sorted.toArray(); for (int i = 0; i < elementos.longitud; i++) {

elemento HistogramElement final = ( HistogramElement) elementos[i]; if (element.getCount() < valor) {

this.histogram.remove(element.getWord()); this.sorted.remove(elemento);

}

}

}

public void removeCommon(final WordHistogram other) { for (elemento histogramelemento final :

other.getSorted()) {

final HistogramElement e = this.get(element.getWord());

if (e == null) {

continuar;

}

this.sorted.remove(e); this.histogram.remove(element.getWord());

}

}

public void removePercent( doble por cientofinal) {

int countdown = (int) (this.sorted.size() \* (1 - porcentaje));

final Object[] elements = this.sorted.toArray(); for (int i = 0; i < elementos.longitud; i++) {

cuenta atrás--;

if (cuenta atrás < 0) {

elemento HistogramElement final = (HistogramElement)elementos[i]; this.histogram.remove(element.getWord()); this.sorted.remove(elemento);

}

}

}

public void trim(final int size) {

final Object[] elements = this.sorted.toArray(); for (int i = 0; i < elementos.longitud; i++) {

if (i >= tamaño) {

elemento HistogramElement final = (HistogramElement)elementos[i]; this.histogram.remove(element.getWord()); this.sorted.remove(elemento);

}

}

}

}

Las palabras individuales del histograma se almacenan en **HistogramElement**

clase. Una colección de objetos **HistogramElement** se almacena dentro de la clase **WordHistogram.** La clase **HistogramElement** se muestra en el listado 13.10.

### Listado 13.10: Elementos del histograma (HistogramElement.java)

paquete com.heatonresearch.book.introneuralnet.ch13.train; importar java.io.Serializable;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* HistogramElement: Un elemento en el histograma. Este es el
* palabra, y el número de veces que surgió en cualquiera de los buenos
* o mal conjunto.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

la clase pública HistogramElement implementa los > Comparable<HistogramElement,

Serializable {

/\*\*

* + Identificador de serie para esta clase.

\*/

private static final long serialVersionUID =

-5100257451318427243L

private Palabra de cadena; private int count;

public HistogramElement(final String word, final int count) { this.word = word.toLowerCase();

this.count = recuento;

}

public int compareTo(final HistogramElement o) { final int result = o.getCount() - getCount(); if (resultado == 0) {

devuelve this.getWord(). compareTo(o.getWord());

} else {

resultado devuelto;

}

}

/\*\*

* + @devolver el recuento

\*/

public int getCount() { return this.count;

}

/\*\*

* + @devolver la palabra

\*/

public String getWord() { return this.word;

}

public void increase() { this.count++;

}

/\*\*

* + @param recuento
  + El recuento que se va a establecer

\*/

public void setCount(final int count) { this.count = count;

}

/\*\*

* + @palabra param
  + La palabra que se va a establecer

\*/

public void setWord(última palabra de cadena) { this.word = word.toLowerCase();

}

}

La clase **HistogramElement** es muy sencilla. Almacena una palabra y el número

de las veces que aparece la palabra. Puede ver estos dos valores en la lista anterior. Ad- ditionally, la clase **HistogramElement** implementa el **comparable** inter- cara, que utiliza para la ordenación. Si examina la lista anterior, verá que el método **compareTo** hace que la ordenación se produzca primero por el número de veces que aparece una palabra y, a continuación, alfabéticamente por la propia palabra.

La clase **WordHistogram,** como se ve en el listado 13.9, proporciona muchos servicios. El bot neuronal los usa en los tres modos de operación. Estos servicios incluyen lo siguiente:

* Contar el número de veces que se producen palabras para crear un histograma
* Compactar una oración a un patrón de entrada
* Quitar palabras comunes que también aparecen en otro histograma
* Quitar el porcentaje inferior de palabras
* Calcular el número medio de apariciones de las palabras en el histograma

Una de las funciones más importantes de la clase **WordHistogram** es compactar una oración en un patrón de entrada para la red neuronal. El patrón de entrada para la red neu- ral es simplemente un recuento relativo del número de ocurrencias de las "buenas palabras" más comunes en la oración. El número de palabras buenas expresadas en el patrón de entrada es el número de neuronas de entrada. La neurona de salida única especifica el grado en que la red neuronal cree que la oración contiene un año de nacimiento.

El patrón de entrada se genera dentro del método **compacto** de **WordHistogram**

clase. La firma del método **compact** se muestra aquí:

public double[] compact(final String line)

El resultado es el mismo tamaño que el número de palabras de este histograma. Este tamaño es el mismo que el número de neuronas de entrada en la red neuronal.

final double[] result = new double[this.sorted.size()];

Se utiliza un objeto **StringTokenizer** para analizar la oración en palabras.

final StringTokenizer tok = new StringTokenizer(línea);

Continuamos con el bucle siempre y cuando haya más tokens para leer.

while (tok.hasMoreTokens()) {

La siguiente palabra se obtiene y se convierte a minúsculas.

final Palabra de cadena = tok.nextToken(). toLowerCase();

A continuación, se da el rango de la palabra. Si la palabra está presente, la neurona de entrada se establece en un 0,9 relativamente alto.

final int rank = getRank(palabra); si (rango ! = -1) {

result[rango] = 0,9;

}

}

Este proceso continúa mientras hay más palabras.

resultado devuelto;

}

Una vez completado el proceso, se devuelve la matriz de entrada resultante para la red neuronal. Esta matriz de entrada se usará para crear conjuntos de entrenamiento o para consultar realmente la red neuronal, una vez que se haya entrenado.

## Creación de los conjuntos de entrenamiento

El conjuntode entrenamiento s para la red neuronal se administran mediante el **TrainingSet**

la clase **TrainingSet** se muestra en el listado 13.11.

### Listado 13.11: Gestión de conjuntos de traning (TrainingSet.java)

paquete com.heatonresearch.book.introneuralnet.ch13.train;

importar java.util. ArrayList; importar java.util.List;

/\*\*

* Capítulo 13: Programación de bots y redes neuronales

\*

* TrainingSet: se usa para crear un conjunto de entrenamiento basado en ambos
* frases buenas y malas.

\*

* @autor Jeff Heaton
* @versión 2.1

\*/

public class TrainingSet {

private List<double[]> input = new ArrayList<double[]>(); private List<double[]> ideal = new ArrayList<double[]>();

public void addTrainingSet(final double[] addInput, final dou- ble addIdeal) {

¿El conjunto de entrenamiento ya existe para (final double[] element : this.input) {

if (compare(addInput, element)) { return;

}

}

agregarlo

matriz doble final[] = nuevo doble[1]; array[0] = addIdeal; this.input.add(addInput); this.ideal.add(matriz);

}

private boolean compare(final double[] d1, final double[] d2)

{

resultado booleano = true;

for (int i = 0; i < d1.longitud; i++) {

if (Math.abs(d1[i] - d2[i]) > 0.000001) { result = false;

}

}

resultado devuelto;

}

/\*\*

* + @devolver el ideal

\*/

public List<double[]> getIdeal() { return this.ideal;

}

/\*\*

* + @devolver la entrada

\*/

public List<double[]> getInput() { return this.input;

}

/\*\*

* + @param ideal
  + el ideal para establecer

\*/

public void setIdeal(final List<double[]> ideal) { this.ideal = ideal;

}

/\*\*

* + @entrada param
  + la entrada que se va a establecer

\*/

public void setInput(final List<double[]> input) { this.input = input;

}

}

Al igual que cualquier otro conjunto de entrenamiento supervisado en este libro, el conjunto de entrenamiento de bots neuronales usa

una matriz de entrada y una matriz ideal. El trabajo más importante de la clase **TrainingSet** es permitir que se agreguen nuevos conjuntos de entrenamiento sin introducir duplicados. Para ello, utiliza el método **addTrainingSet.** La firma del método **addTrainingSet** se muestra aquí:

public void addTrainingSet(final double[] addInput, final double addIdeal)

Se pasan dos argumentos al método **addTrainingSet.** El primero, **addInput**, especifica la matriz deentrada. El segundo, **addIdeal**, especifica los valores ideales para el patrón de entrada especificado.

Comenzamos comprobando si el conjunto de entrenamiento ya existe. Para ello, recorremos todos los elementos.

¿El conjunto de formación ya existe para (final double[] elemento : this.input) {

Si el conjunto de entrenamiento ya está presente, simplemente lo devolvemos e ignoramos.

if (compare(addInput, element)) { return;

}

}

Si no existe, lo añadimos a la lista de conjuntos que ya hemos acumulado.

agregarlo

matriz doble final[] = nuevo doble[1]; array[0] = addIdeal; this.input.add(addInput); this.ideal.add(matriz);

Una vez que se han agregado todos los conjuntos de entrenamiento, se puede entrenar la red neuronal.

## Entrenamiento de la red neuronal

La retropropagación se utiliza para entrenar la red neuronal. Este procedimiento es esencialmente el mismo que todos los demás ejemplos de retropropagación en este libro. El entrenamiento se produce en el método **trainNetworkBackprop** de la clase **TrainBot.** La firma para el método **trainNetworkBackprop** se muestra aquí:

privado vacío trainNetworkBackprop()

En primer lugar, se crea un objeto de entrenamiento. Debido a que esta red neuronal utiliza una función de activación sigmoidal, una tasa de aprendizaje relativamente alta y un alto momento pueden ser speci- fied. Para obtener más información sobre la tasa de aprendizaje y el impulso, consulte el capítulo 5.

tren final = nueva Retropropagación(this.network, this.input, this.ideal, 0.7, 0.7);

La época que comienza es 1.

int época = 1;

Comenzamos un ciclo de entrenamiento.

hacer {

Se realiza cada época del entrenamiento y se informa el error en cada época.

train.iteration();

System.out.println("Backprop:Iteración #" + época + " Error:"

+ train.getError());

época++;

} while ((train.getError() > Config.ACCEPTABLE\_ERROR));

}

El bucle continúa mientras estemos por encima del nivel de error aceptable. Una vez completado el entrenamiento, se guarda la red neuronal, como se ha tratado anteriormente en este capítulo.

# Consulta del bot neuronal

Una vez que el bot ha sido entrenado, se puede usar para buscar el año de nacimiento de una persona famosa. El bot principal para realizar esta función está contenido en el paquete llamado **born**. Para ejecutar el bot, se ejecuta la clase **YearBornBot.** La clase **YearBornBot** es shpropia del Listado 13.12.

### Listado 13.12: Clase WhenBornBot (WhenBornBot.java)

paquete com.heatonresearch.book.introneuralnet.ch13.born;

importar java.io.IOException; importar java.net.URL;

importar java.util. Colección; importar java.util.HashMap; importar java.util.Map;

importar java.util. Conjunto;

importar com.heatonresearch.book.introneuralnet.ch13.Config;

importar com.heatonresearch.book.introneuralnet.ch13.ScanReportable; importar com.heatonresearch.book.introneuralnet.ch13.Text;

importar com.heatonresearch.book.introneuralnet.ch13.train.

WordHistogram;

importar com.heatonresearch.book.introneuralnet.common.YahooSearch; importar com.heatonresearch.book.introneuralnet.neural.feedforward. FeedforwardNetwork;

importar com.heatonresearch.book.introneuralnet.neural.util.

SerializeObject;

/\*\*

* Ejemplo de bot.

\*

* @autor Jeff Heaton
* @versión 1.1

\*/

public class YearBornBot implements ScanReportable { public static final boolean LOG = true;

/\*

* + Objeto de búsqueda que se va a utilizar.

\*/

búsqueda estática de YahooSearch;

/\*\*

* + El método main procesa los argumentos de la línea de comandos
  + y luego llama
  + método de proceso para determinar el año de nacimiento.

\*

* + @param args
  + La persona que se va a escanear.

\*

\*/

public static void main(final String args[]) {

if (args.length < 1) { System.out. println("YearBornBot [Famoso

Persona]"); } else

{

probar {

final YearBornBot cuando = nuevo YearBornBot();

when.process(args[0]); when.process("Bill Gates");

} catch (excepción final e) { e.printStackTrace();

}

}

}

red privada final FeedforwardNetwork; histograma final privado de WordHistogram;

/\*

* + Este mapa almacena una asignación entre un año y cuántos
  + veces ese año
  + ha surgido como un año de nacimiento potencial.

\*/

private final Map<Integer, Integer> results = new HashMap<Integer, Integer>();

public YearBornBot() produce IOException, ClassNotFoundException {

this.network = (FeedforwardNetwork) SerializeObject

.load(Config.FILENAME\_WHENBORN\_NET); this.histogram = (WordHistogram) SerializeObject

.load(Config.FILENAME\_HISTOGRAM);

}

/\*\*

* + Obtenga el año de nacimiento que ocurrió el mayor número de veces.

\*

* + @volver El año de nacimiento que se produjo el más grande
  + número de veces.

\*/

public int getResult() { int result = -1; int maxCount = 0;

final Set<Integer> set = this.results.keySet(); for (final int year : set) {

final int count = this.results.get(año); if (count > maxCount) {

resultado = año; maxCount = recuento;

}

}

resultado devuelto;

}

/\*\*

* + @año param

\*/

private void increaseYear(finalint year) { Integer count = this.results.get(año); if (count == null) {

count = new Integer(1);

} else {

count = new Integer(count.intValue() + 1);

}

this.results.put(año, recuento);

}

/\*\*

* + Se llama a este método para determinar el año de nacimiento para
  + una persona. Obtiene 100 páginas web que Yahoo devuelve
  + para esa persona. A continuación, se busca en cada una de estas páginas
  + para el año de nacimiento de esa persona. Que cada año
  + se selecciona el mayor número de veces se selecciona como
  + el año de nacimiento.

\*

* + @ nombre param
  + El nombre de la persona que está viendo
  + el año del nacimiento para.
  + @lanza IOException
  + Se produce si se produce un error de comunicación.

\*/

public void process(final String name) throws IOException { search = new YahooSearch();

if (YearBornBot.LOG) { System.out.println(

"Obtener resultados de búsqueda de Yahoo." );

}

final Collection<URL> c = search.search(nombre); int i = 0;

if (YearBornBot.LOG) { System.out.println(

"Escaneando URL de Yahoo.");

}

for (URL final u : c) { try {

i++;

Text.checkURL(this, u, null);

} catch (final IOException e) {

}

}

final int resultYear = getResult(); if (resultYear == -1) {

System.out.println("No se pudo determinar cuándo "

+ nombre

+ " nació." );

} else {

System.out.println(nombre + " nació en "

+ resultYear);

}

}

public void receiveBadSentence(oración de cadena final) {

Código auxiliar de método generado automáticamente de TODO

}

public void receiveGoodSentence(oración de cadena final) {

if (this.histogram.count(oración) >= Config.MINIMUM\_WORDS\_PRESENT) {

final double compact[] = this.histogram.compact(oración);

final int year = Text.extractYear(oración);

salida doble final[] = this.network.computeOutputs(compacto);

if (output[0] > 0.8) { increaseYear(año); System.out.println(año + "-"

+ output[0] + ":" + oración);

}

}

}

}

El primer método al que llama el método **main** es el método process. Este método

procesa la solicitud para determinar el año de nacimiento de la persona famosa. El método **de proceso** se trata en la sección siguiente. Otros métodos llamados por el método de **proceso** son covered en secciones posteriores de este capítulo.

**Procesamiento de la** **solicitud**

El método **de proceso** acepta el nombre de la persona famosa e intenta determinar el año de nacimiento de esa persona famosa. La firma para el método de **proceso** se muestra aquí:

public void process(final String name) produce IOException

El método **de proceso** comienza realizando una búsqueda de Yahoo en el **nombre** speci- fied.

buscar = nuevo YahooSearch();

if (YearBornBot.LOG) {

System.out.println("Obtener resultados de búsqueda de Yahoo." );

}

Los resultados de la búsqueda de Yahoo se almacenan en el objeto **c.**

final Collection<URL> c = search.search(nombre); int i = 0;

if (YearBornBot.LOG) {

System.out.println("Escaneando URL de Yahoo.");

}

Recorremos toda la lista y procesamos cada URL. A medida que se procesa cada dirección URL, las oraciones que contienen años de nacimiento se devuelven al método **receiveGoodSentence.** El método **receiveGoodSentence** se tratará en la siguiente sección.

for (URL final u : c) { try {

i++;

Text.checkURL(this, u, null);

Se omiten todos los errores. Llegamos a un gran número de sitios web y algunos están seguros de estar abajo.

} catch (final IOException e) {

}

}

A medida que encontramos años con una alta probabilidad de ser el año de nacimiento, contamos el número de años.

Ber de veces que ocurren. El método **getResult** devuelve el año de nacimiento que es el candidato más probable. El método **getResult** se trata más adelante en este capítulo.

final int resultYear = getResult();

Se devuelve uno negativo si no se encuentra un año de nacimiento probable.

if (resultYear == -1) {

System.out.println("No se pudo determinar cuándo " + nombre

+ " nació." );

De lo contrario, se encuentra el año de nacimiento más probable.

} else {

System.out.println(nombre + " nació en " + resultYear);

}

El método **de proceso** hizo uso de varios otros métodos a medida que se ejecutaba. La siguiente

dos secciones cubrirán estos métodos.

## Recibir sentencias

Al igual que el proceso de recopilación utilizó la clase **Text** para analizar sen- tences individuales, también lo hace el proceso de consulta. A medida que la clase **Text** analiza los datos, se llama al método **receiveGoodSentence** cada vez que se identifica una frase. La firma para el **receiveGoodSentence** se muestra aquí:

public void receiveGoodSentence(oraciónfinal de la cadena)

En primer lugar, comprobamos si la oración encontrada tiene el número mínimo de palabras reconocidas. Si no lo hace, entonces la oración es ignorar.

if (this.histogram.count(oración) >= Config.MINIMUM\_WORDS\_PRESENT)

{

La oración se compacta en una matriz de entrada para la red neuronal. final double compact[] = this.histogram.compact(oración); El año de nacimiento potencial se obtiene de la clase **Text.**

final int year = Text.extractYear(oración);

La salida de la red neuronal se devuelve utilizando la matriz de entrada generada

de la oración actual.

salida doble final[] = this.network.computeOutputs(compacto);

Si la oración especificada causa una salida mayor que 0.8, entonces hay una probabilidad decente de que este sea un año de nacimiento. El recuento se incrementa para este año y se informa el año.

if (output[0] > 0.8) { increaseYear(año);

System.out.println(año + "-" + salida[0] + ":" + oración);

}

}

## Encontrar el mejor candidato para el año de nacimiento

Una vez actualizados todos los recuentos de años, se llama al método **getResult** para determinar qué año tiene el recuento más alto. Este es el año que se propondrá como año de nacimiento para esta persona. La firma del método **getResult** se muestra aquí:

public int getResult()

El **resultado** se establece en un valor predeterminado de -1. Si no se encuentra ningún año de nacimiento potencial, se devuelve este valor -1. La variable **maxCount** se utiliza para realizar un seguimiento de qué año tiene el recuento máximo.

int resultado = -1; int maxCount = 0;

Todos los años identificados como años potenciales del nacimiento se repasan.

final Set<Integer> set = this.results.keySet(); for (final int year : set) {

Se obtiene el recuento para el año especificado.

final int count = this.results.get(año); if (count > maxCount) {

resultado = año; maxCount = recuento;

}

}

Continuamos haciendo un bucle y actualizando el resultado a medida que se encuentran mejores candidatos.

resultado devuelto;

}

Este proceso continúa hasta que se ha seleccionado al mejor candidato.

# Resumen del capítulo

Los bots son programas informáticos que realizan tareas repetitivas. A menudo, un bot hace uso de un protocolo HTTP para acceder a sitios web de la misma manera que los humanos. Dado que un bot generalmente tiene acceso a una gran cantidad de datos, las redes neuronales pueden ser útiles para ayudar al bot a comprender los datos entrantes.

Este capítulo presentó una red neuronal que intenta determinar el año de nacimiento de una persona famosa. Esta red neuronal se divide en tres modos distintos de operación. El primer modo utiliza un bot encadenado para recopilar mucha información sobre el año de nacimiento de personas famosas especificadas. Usando estas personas famosas y sus años de nacimiento, el bot se entrena en el segundo modo. El modo final permite al usuario consultar la red y determinar los años de nacimiento de otras personas famosas.

Este libro ha cubierto un adormecimientoer de diferentes tipos de redes neuronales y métodos para su uso. A pesar de todo lo que son capaces de hacer, las redes neuronales todavía se quedan considerablemente cortas para lograr la capacidad de razonar como un cerebro humano. El siguiente capítulo analizará el estado actual de la investigación de redes neuronales y hacia dónde se dirigen las redes neuronales en el futuro.

# vocabulario

Histograma de bot

Análisis del protocolo HTTP

# Preguntas para revisión

1. ¿Cómo sugiere este capítulo transformar una oración en un patrón de entrada para la red neuronal?
2. Aparte del método descrito en este capítulo, ¿de qué otra manera podría transformarse una oración en entrada para una red neuronal? ¿Es necesario que cada oración produzca el mismo número de entradas para la red neuronal?
3. ¿Por qué la función de activación sigmoidal fue una buena opción para una función deactivación- ción para el bot neuronal?
4. ¿Qué es el análisis de texto y cómo es útil?
5. Si desea expandir este bot para localizar también el nombre del cónyuge de la persona famosa, ¿cómo se podría hacer esto?