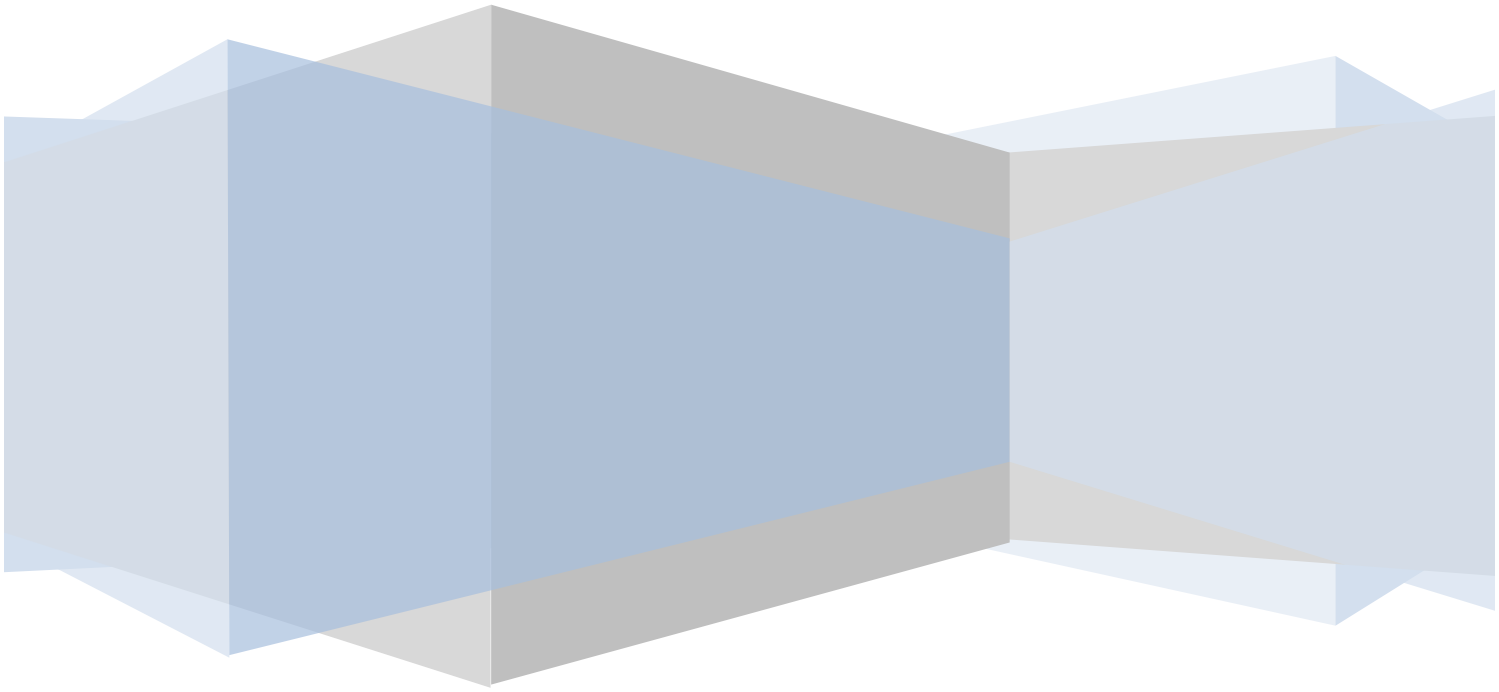
Universidad de Zaragoza

Práctica 5:

Hadoop y HBase

Adrian Casans (590114) Sergio Pedrero (627669) Diego Sánchez (628279) Cristian Simón (611487)



ÍNDICE

Esfuerzos invertidos:

Tabla que recoge un desglose con las tareas y esfuerzos realizados por cada uno de los miembros del grupo junto con el tiempo invertido:

Tarea	Encargados	Horas
Explicación Instalación Hadoop	Cristian	1'5
Explicación Instalación HBase	Cristian	1'5
Inserción datos HBase	Diego / Cristian	9 / 4'5
Modelo predictivo con MapReduce	Sergio / Adrian	20/ 4
Clase Test y pruebas ligadas	Adrian	2
Memoria	Todos	1'5

Objetivos:

El objetivo del desarrollo de la practica consistirá en:

- Familiarizarse con el paradigma paraleloMapReduce y su implementación Hadoop.
- Familiarizarse con la base de datos NoSQL HBase.
- Aprender nociones básicas del problema de clasificación en minería de datos.
- Realizar una pequeña aplicación de ejemplo.

1.Instalación de Hadoop y HBase.

1.1 Hadoop Instalación

Para la instalación de Hadoop debemos tener previamente instalado el JDK de Java. Añadiremos un nuevo grupo al que denominaremos "hadoop" y añadiremos un usuario a ese grupo:

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

Instalaremos SSH (SSH y SSHD), para ello ejecutaremos:

```
sudo apt-get install ssh
```

Ahora configuraremos nuestro SSH ya que Hadoop requiere el acceso SSH para administrar sus nodos, para ello crearemos un certificado SSH:

```
su hduser
ssh-keygen -t rsa -P ""
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Instalamos Hadoop (version 2.6), para ello:

```
wget     http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
tar xvzf hadoop-2.6.0.tar.gz
```

Movemos la instalación de Hadoop a **/usr/local/hadoop**, para ello primero deberemos poner el usuario hduser en el fichero sudoers, por lo que lo añadiremos:

```
su
sudo adduser hduser sudo
sudo su hduser
```

Y dentro de la carpeta /hadoop-2.6.0:

```
sudo mv * /usr/local/hadoop
sudo chown -R hduser:hadoop /usr/local/hadoop
```

Ahora procederemos a la configuración de Hadoop, para ello modificaremos **/.bashrc**

```
vi ~/.bashrc
```

añadiendo:

```
#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

```
source ~/.bashrc
```

Definiremos el **JAVA_HOME** modificando **hadoop-env.sh**:

```
vi /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

añadimos:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Modificaremos la configuración cuando Hadoop se lanza:

```
sudo mkdir -p /app/hadoop/tmp
sudo chown hduser:hadoop /app/hadoop/tmp
```

para ello modificaremos el fichero **core-site.xml**:

```
vi /usr/local/hadoop/etc/hadoop/core-site.xml
```

añadiendo entre **<configuration></configuration>:**

```
<configuration>
 <property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
 </property>

 <property>
  <name>fs.default.name</name>
```

```
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
 </property>
</configuration>
```

Por defecto **/usr/local/hadoop/etc/hadoop/ contiene el fichero /usr/local/hadoop/etc/hadoop/mapred-site.xml.template** que tendremos que renombrar con el nombre **mapred-site.xml**

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

Ahora lo modificaremos añadiendo entre <configuration></configuration>:

```
<configuration>
 <property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
 </property>
</configuration>
```

Ahora modificaremos el fichero **hdfs-site.xml**, ya que es necesario configurarlo para cada host que se utilice y en el especificaremos los directorios que serán utilizados.

Antes de editarlo necesitamos crear 2 directorios, **NameNode** y **DataNode**:

```
sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
```

```
sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
s
```

editamos:

```
vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

y añadimos entre **<configuration></configuration>:**

```
<configuration>
 <property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
 </property>
</configuration>
```

Una vez hayamos hecho todo esto proceremos a formatear el sistema de archivos de Hadoop para que podamos comenzar a utilizarlo, para ello:

```
hadoop namenode -format
```

1.2 Iniciar Hadoop

Ya tenemos Hadoop, precederemos a iniciarlo:

```
cd /usr/local/hadoop/sbin
```

```
sudo su hduser
start-all.sh
```

para comprobar que está en marcha:

```
jps
```

obtendremos:

```
9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode
```

1.3 Detener Hadoop

Para detener Hadoop sera lo mismo pero esta vez ejecutaremos:

```
cd /usr/local/hadoop/sbin
sudo su hduser
stop-all.sh
```

1.4 Web UI Hadoop

Teniendo Hadoop activado podemos acceder a su Web UI:

http://localhost:50070/

2.1 HBase Instalación

Para la instalación de Hadoop debemos descargarnoslo, utilizaremos la version 0.98.12. Una vez lo tengamos descargado lo descomprimimos:

```
tar xvzf hbase-0.98.12-hadoop2-bin.tar.gz
```

y lo movemos a la carpeta /usr

```
sudo mv hbase-0.98.12-hadoop2 /usr/local/hbase/
```

Añadimos en el **bashrc** las siguientes lineas:

```
export HBASE_HOME=/usr/local/hbase/hbase-0.98.12-hadoop2
export PATH=$PATH:$HBASE_HOME/bin
```

y lo ejecutamos:

```
source ~/.bashrc
```

Definimos el Home de HBase, para ello modificamos **hbase-env.sh** ubicado en **/usr/local/hbase/conf/**:

```
vi conf/hbase-env.sh
```

y añadimos:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HBASE_REGIONSERVERS=${HBASE_HOME}/conf/regionservers
export HBASE_MANAGES_ZK=true
```

hora editamos el **core-site.xml**, ubicado en **/usr/local/hadoop/etc/hadoop/** y añadimos entre **<configuration></configuration>**

```
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://localhost:54310/hbase</value>
  </property>

  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>

  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>localhost</value>
  </property>

  <property>
```

```
    <name>dfs.replication</name>
    <value>1</value>
  </property>

  <property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2181</value>
  </property>

  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/hduser/hbase/zookeeper</value>
  </property>
```

2.2 Iniciar HBase

Ya tenemos HBase, precederemos a iniciarlo, para ello vamos a **/usr/local/hbase/bin** y ejecutamos:

```
./start-hbase.sh
```

Comprobamos que se esta ejectuando con:

```
jps
```

y deberemos ver "HMaster":

```
17874 DataNode
20199 HMaster
18048 SecondaryNameNode
20129 HQuorumPeer
20374 HRegionServer
20480 Jps
18182 ResourceManager
18488 NodeManager
17733 NameNode
```

2.3 HBase Shell

Para iniciar el shell de HBase:

```
hbase shell
```

2.4 Ejemplo de uso

Ejemplo de **creacion** de una tabla:

hbase(main):008:0> create 'practica5', 'user', 'datos'

0 row(s) in 0.9270 seconds

=> Hbase::Table - practica5

hbase(main):009:0> list

TABLE

practica5

1 row(s) in 0.0240 seconds

Añadir valores a esa tabla:

hbase(main):010:0> put 'practica5', 'row1', 'user:uid', '1SCOGBVSVJ'

0 row(s) in 0.4410 seconds

Ver esos valores:

hbase(main):017:0> get 'practica5', 'row1'

COLUMN	*CELL*
datos:caract1	*timestamp=1464000191364, value=0.0*
datos:caract2	*timestamp=1464000200499, value=0.0*
datos:caract3	*timestamp=1464000206937, value=0.0*
datos:caract4	*timestamp=1464000215150, value=1.0*
user:group	*timestamp=1464000131961, value=A*
user:uid	*timestamp=1464000104361, value=1SCOGBVSVJ*

Eliminar tabla:

disable 'practica5'

drop 'practica5'

2.Insercion de datos en HBase.

Para la inserción de datos en Hbase se ha utilizado Eclipse y el plugin de Hadoop para Eclipse. Una vez instalado se ha creado un nuevo proyecto "Map reduce" y se han importado las librerías de HBase (/usr/local/hbase/lib) al proyecto creado.

Para trabajar con el CSV se ha necesitado la importación de la libreria OpenCSV, esta librería nos proporciona métodos para leer y escribir este tipo de ficheros. Posteriormente, analizando el contenido del fichero, ya que se trata de un CSV(comma separated values) se podría haber procedido con un Scanner al fichero y analizándolo línea a línea definiendo como separador para la lectura de cada campo el string “,”.Así nos evitaríamos trabajar con la librería OpenCSV, aunque al final se ha decidido optar por esta opción.

Clase en Java:

```
import java.io.IOException;


import java.io.FileReader;

import com.opencsv.CSVReader;


import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.HColumnDescriptor;

import org.apache.hadoop.hbase.HTableDescriptor;

import org.apache.hadoop.hbase.TableExistsException;

import org.apache.hadoop.hbase.client.HBaseAdmin;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.util.Bytes;


public class ImportFile{


      public static void main(String[] args) throws IOException

      {

            //CREA LA TABLA SI NO EXISTE

            try{

                  /*Crea una instancia HBaseAdmin con la instancia de
la configuracion*/

                  Configuration config = HBaseConfiguration.create();

                  HBaseAdmin hbase_admin = new HBaseAdmin( config );

                  /*Crear descriptor de tabla*/

                  HTableDescriptor htable = new
HTableDescriptor("practica5");

                  /*Crear descriptor de la familia de columnas*/

                  HColumnDescriptor user = new
HColumnDescriptor("user");
```

```
                HColumnDescriptor datos = new
HColumnDescriptor("datos");
                /*Añadir familias de columnas a la tabla*/
                htable.addFamily(user);
                htable.addFamily(datos);
                System.out.println( "Connecting..." );
                System.out.println( "Creating Table..." );
                hbase_admin.createTable( htable );
                System.out.println("Done!");
          } catch(TableExistsException e){
                System.out.println("Ya existe la tabla practica5.");
          }

          //CONECTA CON LA TABLA
          org.apache.hadoop.conf.Configuration config =
HBaseConfiguration.create();
          HTable table = new HTable(config, "practica5");

          //LEE EL FICHERO
          try {
                //csv file containing data
                String strFile = "data.csv";
                CSVReader reader = new CSVReader(new
FileReader(strFile));
                String [] nextLine;
                int lineNumber = 0;
                //recorre cada linea
                while ((nextLine = reader.readNext()) != null) {

                      lineNumber++;
                      Put p = new
Put(Bytes.toBytes("row"+lineNumber));
```

```
                        System.out.println("Line # " + lineNumber);


                        p.add(Bytes.toBytes("user"),
Bytes.toBytes("uid"),Bytes.toBytes(nextLine[0])); //código unico de
usuario


      p.add(Bytes.toBytes("user"),Bytes.toBytes("grupo"),Bytes.toBytes
(nextLine[1])); //grupo A o B



                        int aux;

                        // nextLine[] is an array of values from the
line

                        // inserta cada característica

                        for (int i=2; i<nextLine.length; i++) {

                              aux = i - 1;

                              if(Double.parseDouble(nextLine[i])==1.0){

                                    p.add(Bytes.toBytes("datos"),
Bytes.toBytes(""+aux),Bytes.toBytes(nextLine[i]));

                              }

                        }

                        table.put(p);

                  }

                  reader.close();

            } catch(Exception e) {

                  e.printStackTrace();

            }

      }


}
```

3. Modelo predictivo con MapReduce

El modelo predictivo que vamos a utilizar está asado en la Regresión Logística.

La regresión logística es un método de clasificación supervisado, paramétrico y basado en técnicas estadísticas.

¿Modelo? Representación de la realidad compleja mediante alguna formulación matemática. En este caso ¿ Podemos encontrar 1601 valores reales $\theta_0,...\theta_{1600}$, de forma que dado un dato cualquiera podemos clasificarlo según la función f(x)?(guión función 4.1)

¿Paramétrico? Depende de unos parámetros θ_i desconocidos .

¿Supervisado? Se utilizan un conjunto de datos que sabemos bien clasificados(en nuestro caso los datos almanceados en HBase del fichero data.csv) para hallar los valores de los parámetros θ, con la esperazana de que si el modelo claisfica bien los datos conocidos será capaz de clasificar también nuevos datos.

El algoritmo que nuestro programa debe realizar en notación genérica seria el siguiente :

$\theta = (0,0,...,0)$

while (no se cumpla criterio de convergencia) {

/ / *vector de 1601 posiciones , $grad_j$ es la*

grad = (0,0,...,0)

for j←0**to**1600{

$$grad_j = \mathrm{E}^{n}_{i=1}(g(\theta \mathbf{x}^i) - y^i) * x^i_j$$

}

$\theta = \theta - \alpha * \mathbf{grad}$

}

El único sitio en el que merece la pena aplicar map/reduce para ajustar el modelo predictivo(hallar los valores de θ es el sumatorio :

$$grad_j = \mathrm{E}^{\,n}_{i=1}(g(\theta \mathbf{x}^i) - y^i) * x^i_j$$

A continuación se describen las clases Java utilizadas mediante la API de Hadoop y HBase para realizar lo descrito anteriormente(está todo comentado) :

Logistic.java

```
import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableMapReduceUtil;

import org.apache.hadoop.hbase.mapreduce.TableMapper;

import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class Logistic {


        //Clase mapper


        public static class DataMapper extends TableMapper<IntWritable,

        DoubleWritable> /* Tipo de valor de salida(valor de j,valorIntermedio)*/

        {
```

```
		private double[] tetha = new double[1601]; // Vector para el calculo de los 1601 valores que permiten aproximar la clasificacion de un dato
		private double[] aux = new double[1601]; /*Estructura auxiliar que se le pasa al cleanup*/

		@Override
		public void setup(Context context) {
			/*Devuelve la configuracion del contexto*/
			Configuration c = context.getConfiguration();
			/*Asigna las variables del contexto(MapReduce) al vector result*/
			for(int i=0;i<tetha.length;i++){
				tetha[i]= c.getDouble("tetha"+i,0.0);
			}
		}

		/**
		 * @param row Valor actual de la clave de cada fila(registro)
		 * @param value Las columnas
		 * @param context El contexto actual
		 * @throws IOException
		 * @throws InterruptedException
		 * @see org.apache.hadoop.mapreduce.Mapper#map(KEYIN, VALUEIN,
		 * org.apache.hadoop.mapreduce.Mapper.Context)
		 */
		@Override
		public void map(ImmutableBytesWritable row,Result value,Context context){
			double y = 1.0;
			/*Valor de y(entero) despues de comprobar a que grupo pertenece el dato perteneciente a row*/
```

```
                    if(Bytes.toString(value.getValue(Bytes.toBytes("user"),
Bytes.toBytes("grupo"))).equals("A")){
                        y = 0.0;
                    }

                    /*Calculo de la funcion g(0*Xi)*/

                    double z = 0.0;
                    double g = 0.0;

                    /*Calculamos el valor del dato xi en este caso llamado [x]*/
                    double[] x = new double[1601];
                    x[0] = 1.0;

                NavigableMap<byte[], byte[]> familyMap =
value.getFamilyMap(Bytes.toBytes("datos"));
                 for(byte[] bQunitifer : familyMap.keySet())
                 {
                   x[Integer.parseInt(Bytes.toString(bQunitifer))]  = 1.0;

                 }
                    /*Calculo de tetha*Xi */
                    for(int i=0;i<x.length;i++){
                        z +=  tetha[i]* x[i];
                    }
                    /*Calculo de g(0x)*/
                    g = (1/1+Math.exp(-z));
                    /*Calculo de g(0x)-y*/
```

```
                    g = g-y;


                    /*Y multiplicamos por cada xij*/
                    for(int j=0;j<aux.length;j++){
                        aux[j] += g*x[j];
                    }
                }
                /* El cleanup le pasa al reducer el par <j,'valorIntermedioParaJ'> que estaba almacenado en un vector
                 * auxiliar llamado aux */
                @Override
                public void cleanup(Context context) throws IOException, InterruptedException{
                    for(int j=0;j<1601;j++){
                        context.write(new IntWritable(j),new DoubleWritable(aux[j]));
                    }
                }

        }
        // Clase reducer

        public static class DataReducer extends Reducer<
        IntWritable,                    // Tipo de clave de entrada
        DoubleWritable,                 // Tipo de valor de entrada
        IntWritable,                    // Tipo de clave de salida
        DoubleWritable>                         // Tipo de valor de salida
        {
```

```
		public void reduce(IntWritable j ,Iterable<DoubleWritable> values, Context
context)  {
			/*Se acumula en suma el valor de cada resultado intermedio que ha
generado el map*/
			double suma = 0.0;
			for(DoubleWritable val: values ){
				suma += val.get();
			}
			try {
				// Se escribira en el fichero el par
<j,'resultadoTotalParacadaj'>
				context.write(j, new DoubleWritable(suma));
			} catch (IOException | InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

	public static void main(String[] args) throws Exception{
		int convergencia = 0;
		double[] tetha = new double[1601];
		while(convergencia<5){
			Configuration conf = HBaseConfiguration.create();
			for(int i=0;i<tetha.length;i++){
				tetha[i]= conf.getDouble("tetha"+i,0.0);
			}
			Job job = Job.getInstance(conf,"Regresion logistica");
			Scan scan = new Scan();
```

```
                scan.setCaching(500);
                job.setJarByClass(Logistic.class);
                //Definir scan para leer la tabla

                job.setJarByClass(Logistic.class);
                TableMapReduceUtil.initTableMapperJob(
                                "practica5",                    //nombre de la
tabla
                                scan,                                   //
Instancia scan para controlar CF y atributos
                                DataMapper.class,               // Clase
mapper
                                IntWritable.class,              // Tipo de
clave de salida del mapper
                                DoubleWritable.class,           // Tipo de valor de
salida del mapper
                                job);
                job.setReducerClass(DataReducer.class); // clase reducer
                job.setNumReduceTasks(6); // al menos una, ajustar si se requiere
                job.setOutputKeyClass(IntWritable.class);
                job.setOutputValueClass(DoubleWritable.class);

                FileOutputFormat.setOutputPath(job, new Path("out"));

                if (!job.waitForCompletion(true))
                        return;
                for(int i=0;i<tetha.length;i++){
                        tetha[i]= conf.getDouble("tetha"+i,0.0);
                }
```

```
                    Scanner recorrerFichero = new Scanner(new File("out/part-r-
00000"));

                    String linea ="";

                    while(recorrerFichero.hasNextLine()){

                            int j = recorrerFichero.nextInt();

                            double valor = recorrerFichero.nextDouble();

                            tetha[j] = tetha[j] - 0.01*valor;

                            recorrerFichero.nextLine();

                            System.out.println(tetha[j]);

                    }


                    convergencia++;


            }

        }

}
```

3. Pruebas. Problemas y dificultades encontrados.

. *Validation.java*

Ya que se tuvieron problemas diversos con la instalación de HBase y Hadoop y hasta el ultimo momento no se ha podido verificar que el sistema funcionaba correctamente, teniendo que formatear el sistema de ficheros distribuido DFS y habiendo tenido que reinstalar el entorno 4 o 5 veces, adjuntamos un esqueleto de lo que debería contener la clase Validation para las pruebas.

En conclusión, nuestro modelo map/reduce está bien implementado pero por algún motivo relacionado con las versiones utilizadas de Hadoop y HBase, creyendo hasta el prácticamente los dos últimos días que todo estaba correctamente instalado, no era así y hablando con uno de los tutores de la asignatura y corroborándolo este es el resultado de todo lo que conseguimos llegar a hacer funcionar correctamente.

Disculpándonos por esto ultimo y no haber podido solventarlo con mas rapidez esperamos que valoren positivamente esta practica, adjuntamos por ultimo esta clase "ficticia" que creemos tendrá el siguiente contenido

import java.util.*;

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.client.Get;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.hdfs.DFSClient.Conf;

import org.apache.hadoop.io.*;

```
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;




public class Validation {


   public static void main []{
         String strFile2 = "tetha.csv";
         CSVReader tethar = new CSVReader(new FileReader(strFile2));
         Double[] tetha = new Double[1601];
         tetha = tethar.readNext;
         tethar.close();

         String strFile = "dataTest.csv";
         CSVReader reader = new CSVReader(new FileReader(strFile));
         String [] nextLine = new String[1601];
         int lineNumber = 0;

         /**
          * Resultados
          */
         int aBien = 0;
         int aMal = 0;
         int bBien = 0;
         int bMal = 0;
```

```
while ((nextLine = reader.readNext()) != null) {
        lineNumber++;
        String letra = nextLine[1].toString(); //grupo A o B

        int aux = 0;
        double z = 0.0;
        double dato = 0;
        for (int i=2; i<nextLine.length; i++) {
                aux = i - 1;
                dato = Double.parseDouble(nextLine[i]);
                z+=dato*tetha[aux];
        }
        double fx = (1/1+Math.exp(-z));
        if(letra.equals("A")){
                if(fx<0.5){
                        aBien++;
                }
                else bMal++;
        }
        else{
                if(fx<0.5){
                        aMal++;
                }
                else bBien++;
        }
```

```
                System.out.println("A                B");
                System.out.println("A       "+aBien+("    ")+bMal);
                System.out.println("B       "+aMal+("    ")+bBien);
        }
        reader.close();


    }
}
```