Digit Recognizer: Competición en *Kaggle*

Descubrimiento de conocimiento en bases de datos

Universidad Politécnica de Madrid

Cristina Martín Bris

cristina.martin.bris@alumnos.upm.es

Enero 2019

Resumen

El proceso KDD tiene como objetivo descubrir conocimiento que existe implícitamente dentro de los datos. El presente trabajo tiene como objetivo la resolución del problema de reconocimiento de dígitos existente en la plataforma *Kaggle*, mediante la aplicación de procesos de descubrimiento de conocimiento y la participación activa en la competición. Para ello, primeramente, se ha procedido a hacer un primer entendimiento de los datos. Seguidamente se han explorado las diferentes aproximaciones posibles a la solución mediante diferentes algoritmos de clasificación existentes, de los cuales se ha elegido únicamente uno de ellos para trabajar con él durante el resto del proyecto. A continuación, se ha procedido a realizar algunas modificaciones en el conjunto de datos original con el fin de mejorar los resultados obtenidos. Se ha observado que el algoritmo *Random Forest* ha sido uno de los algoritmos que mejores resultados ha obtenido.

Abstract

The KDD process aims to discover knowledge that exists implicitly within the data. The objective of this paper is to solve the digit recognition problem existing in the Kaggle platform, through the application of knowledge discovery processes and active participation in the competition. For this, firstly, we have proceeded to make a first understanding of the data. Next, the different possible approaches to the solution have been explored through different existing classification algorithms, of which only one of them has been chosen to work with it during the rest of the project. Then, some modifications were made to the original data set in order to improve the results obtained. It has been observed that the Random Forest algorithm has been one of the algorithms that has obtained the best results.

I. Contexto del problema

MNIST (base de datos modificada del Instituto Nacional de Estándares y Tecnología) es una gran base de datos de dígitos escritos a mano escritos por estudiantes de secundaria y empleados de la Oficina del Censo de los Estados Unidos publicada en el año 1999 [1].

MNIST contiene un total de 70000 imágenes de dígitos. Estas imágenes se obtuvieron a través del escaneado de documentos, normalizados en tamaño y centrados. Cada imagen tiene un tamaño de veintiocho por veintiocho píxeles. En las imágenes solo aparecen diez dígitos, los cuales se corresponden con los dígitos del cero al nueve.

Comúnmente, este conjunto de datos se utiliza para entrenar sistemas de procesamiento de imágenes para el reconocimiento de patrones.

En el área del análisis de datos, los datos extraíbles de MNIST han sido de gran utilidad para probar los algoritmos de clasificación emergentes en ese momento y hoy en día sigue siendo un recurso fiable para probar las nuevas técnicas de aprendizaje automático. [2].

Son muchas las técnicas que han sido utilizadas en este conjunto de datos y algunos de los resultados que se han obtenido de ellas son satisfactorios, llegando a conseguir en algunos casos 99,73% de correctas clasificaciones [3]. Para ello, los algoritmos de clasificación que han sido utilizados fueron clasificadores lineales, vecinos más próximos, máquinas de soporte vectorial y redes neuronales entre otros.

El objetivo del presente trabajo es participar en la competición abierta de la plataforma Kaggle que se basa en el conjunto de datos anterior.

Kaggle se trata de una plataforma online que permite a los usuarios participar y crear competiciones de análisis de datos. En estas competiciones están publicados los conjuntos de datos necesarios a partir de los cuales se construirán los modelos con los que se participará en la misma. Cada participante de la competición puede subir los resultados del entrenamiento de su modelo. La plataforma dará al usuario, una vez subido su modelo, un porcentaje de buenas clasificaciones y entrará, según este porcentaje, en un ranking en el que están el resto de participantes de la competición.

Para la participación en esta competición, se requerirá aplicar los diferentes pasos del proceso de descubrimiento de conocimiento y la aplicación de diferentes heurísticas para la correcta clasificación del conjunto de datos.

Cada uno de los pasos seguidos durante este proceso se revisarán a continuación.

II. Descripción del conjunto de datos

El conjunto de datos con el que se ha trabajo se ha descargado de la plataforma *Kaggle*. Los datos que se han proporcionado están divididos en tres ficheros.

Nombre fichero	Tamaño	Número de registros
train.csv	73,2 MB	42.000
test.csv	48,7 MB	28.000
sample_submission.csv	0.229492 MB	28.000

Tabla1: Ficheros conjunto de datos

Los ficheros *train.csv* y *test.csv* se corresponden con el conjunto de datos con el que se va a trabajar, aplicando en ellos las diferentes técnicas de clasificación. El uso del conjunto de datos *train.csv* está planteado para el entrenamiento del modelo, en nuestro caso, de clasificación. El fichero *test.csv* se utilizará para probar el rendimiento del modelo entrenado anteriormente.

El fichero *sample_submission.csv* se corresponde con un fichero que presenta el formato requerido del fichero que contiene las clasificaciones finales del modelo. Este fichero debe de ser subido a la plataforma *Kaggle* para participar activamente en la competición.

Como ya se ha planteado anteriormente, las imágenes contenidas en los ficheros *train.*csv y *test.*csv son imágenes de 28x28 píxeles, por lo que una imagen cuenta con un total de 784 píxeles.

El fichero *train.csv* contiene los siguientes atributos:

Atributo	Descripción	Tipo	Valores	Nulos
Label	Atributo de clase. Número dibujado por la persona	Nominal	[0,9]	0
Pixel0	Valor del píxel 0 en la imagen.	Numérico	[0,255]	0
…		…	…	…
Pixel783	Valor del píxel 783 en la imagen.	Numérico	[0,255]	0

Tabla 2: descripción atributos fichero *train.csv*

El fichero *test.csv* contiene los siguientes atributos:

Atributo	Descripción	Tipo	Valores	Nulos
Pixel0	Valor del píxel 0 en la imagen.	Numérico	[0,255]	0
…		…	…	…
Pixel783	Valor del píxel 783 en la imagen.	Numérico	[0,255]	0

Tabla 3: descripción atributos fichero *test.csv*

Debido a que no existen valores nulos ni atributos con valores vacíos, no se ha requerido ninguna técnica de preprocesado de los datos con este fin en ninguna de las etapas.

En el fichero *test.csv* no aparece el atributo "*label"* debido a que es el atributo de clase. El atributo de clase es la clasificación final que se da, en este caso, a la imagen. El objetivo del conjunto de datos *test.csv* es probar el modelo desarrollado previamente y asignarle el atributo de clase que se obtiene del modelo clasificador.

III. Exploración de los datos

Con el fin de tener un conocimiento más profundo del conjunto de datos con el que iba a trabajar, se realizó una primera exploración de los datos

En los ficheros con los que se trabajó, las imágenes estaban representadas a través de una fila donde aparecen todos los valores de cada uno de los píxeles de la imagen. Por este motivo, se realizó una conversión a imagen real donde se pudiera ver la forma de las imágenes reales.

El código desarrollado para esta visualización fue el siguiente [3]:

```
#explorations
library(RColorBrewer)
BNW <- c("white", "black")
CUSTOM_BNW <- colorRampPalette(colors = BNW)

par(mfrow = c(4, 3), pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
images_digits_0_9 <- array(dim = c(10, 28 * 28))
for (digit in 0:9) {
  images_digits_0_9[digit + 1, ] <- apply(train_orig[train_orig[, 1] == digit, -1], 2, sum)
  images_digits_0_9[digit + 1, ] <- images_digits_0_9[digit + 1, ]/max(images_digits_0_9[digit + 1, ]) * 255
  z <- array(images_digits_0_9[digit + 1, ], dim = c(28, 28))
  z <- z[, 28:1]
  image(1:28, 1:28, z, main = digit, col = CUSTOM_BNW(256))
}
```

Del cual se obtuvo el siguiente resultado:

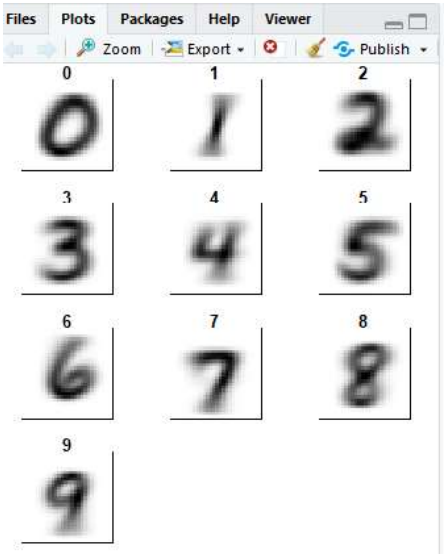

Imagen 1: vista dígitos del dataset

Otra de las exploraciones que se llevó a cabo fue el conocer el porcentaje de representación de cada valor del atributo de clase, con el fin de conocer si el conjunto de datos con el que se estaba trabajando estaba balanceado o no.

El código utilizado para este propósito es el siguiente [3]:

```
CUSTOM_BNW_PLOT <- colorRampPalette(brewer.pal(10, "Set3"))
LabTable <- table(train_orig$label)
par(mfrow = c(1, 1))
percentage <- round(LabTable/sum(LabTable) * 100)
labels <- paste0(row.names(LabTable), " (", percentage, "%) ")
pie(LabTable, labels = labels, col = CUSTOM_BNW_PLOT(10), main = "Percentage of Digits (Training Set)")
```

El resultado obtenido fue el siguiente:

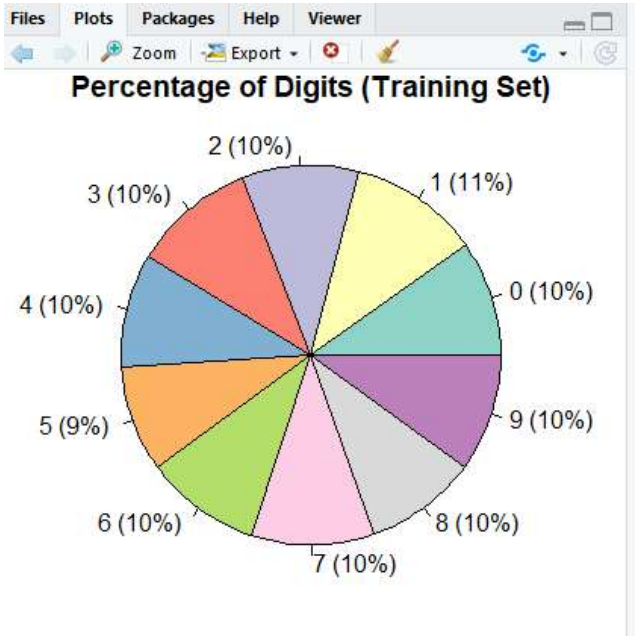

Imagen 2: porcentaje de representación de los dígitos en el dataset

De esta imagen se observa que todos los valores tienen un porcentaje de representación en el conjunto de datos muy similar, por lo que podemos decir que no se requiere de ninguna técnica de balanceamiento de los datos para poder obtener un modelo que pueda dar resultados aceptables.

IV. Exploración de algoritmos de clasificación

Yann LeCun, profesor del Instituto Courant de Ciencias Matemáticas de la Universidad de Nueva York, hizo una revisión sobre los diferentes métodos que se habían aplicado al conjunto de datos con el que se ha trabajado [4]. En esta revisión, existen algunos algoritmos de clasificación clásicos que no están presentes.

Por este motivo, se decidió hacer una primera aproximación para observar con qué algoritmo se obtienen mejores clasificaciones.

Se han elegido algoritmos de clasificación tanto utilizados como no utilizados en la revisión debido a que los resultados de clasificación de los algoritmos pueden verse modificados por la implementación y el software utilizado. Todas las implementaciones de algoritmos se han desarrollado en el lenguaje de programación *R*.

Para esta exploración de los algoritmos se ha utilizado como conjunto de datos de entrenamiento el fichero *train.csv* y como conjunto de datos de test el fichero *test.csv*, ambos ficheros siendo los aportados por la plataforma sin sufrir ninguna modificación.

A continuación, se verá en detalle el rendimiento de cada uno de los algoritmos:

a. Random Forest

El algoritmo *Random Forest* se trata de un algoritmo de aprendizaje supervisado. Este algoritmo crea un bosque a partir de un conjunto de arboles decisión. El resultado final de este algoritmo es la combinación de cada uno de los resultados obtenidos en cada árbol del bosque.

El porcentaje de aciertos en clasificación para este algoritmo fue 92,75%.

b. Naïve Bayes

Naïve Bayes es un algoritmo de aprendizaje supervidado. Este algoritmo clasifica los elementos del conjunto de datos a través de una clasificación probabilística basada en una simplificación del teorema de Bayes [5].

El porcentaje de aciertos en la clasificación para este algoritmo fue 52,642%, lo que se puede considerar un resultado poco satisfactorio.

c. SVM

El algoritmo SVM se trata también de un algoritmo de aprendizaje supervisado. Una SVM separa los n valores del atributo de clase en n espacios lo más amplios posibles separándolos mediante un hiperplano, los cuales se denominan vector soporte [6].

Una vez entrenado el modelo, los datos de entrada pueden ser clasificadas en cualquiera de las n clases existentes.

El porcentaje de aciertos en la clasificación para este algoritmo fue 91,44%, lo que se puede considerar un resultado satisfactorio.

d. Decision Tree

Los arboles de decisión también se consideran algoritmos de aprendizaje supervisado. En este algoritmo, se fabrica un árbol de decisión a través de divisiones lógicas en los datos (de forma muy parecida a las basadas en reglas).

El porcentaje de aciertos en la clasificación para este algoritmo fue 74,3%.

f. Red neuronal

Una red neuronal se trata de un modelo computacional que trata de recrear el aprendizaje de las personas. Se basa en un gran conjunto de nodos, los cuales recrean las unidades neuronales. El comportamiento de este modelo es muy parecido al comportamiento cerebral. El conjunto de datos de entrada atraviesa la red neuronal y produce unos valores de salida, los cuales serán el resultado de la clasificación.

El porcentaje de aciertos en la clasificación para este algoritmo fue 90,757%.

De los resultados obtenidos mediante esta primera exploración, se observó que el algoritmo de clasificación con el que mejores resultados se obtenían era el algoritmo *Random Forest*, por lo que se decidió trabajar con este algoritmo durante el resto de proyecto, intentando mejorar el porcentaje de buenas clasificaciones.

Algunas de las características del modelo construido a partir de este algoritmo son las siguientes: el modelo ha sido construido a partir de los datos de entrenamiento (fichero *train.csv)* y testeado a través del conjunto de datos de prueba (*test.csv)*. El bosque resultante está formado por un total de 20 árboles.

Los detalles del modelo desarrollado son los siguientes:

Comentado [CMB1]: Dar más detalles sobre randomforest: fundamento teorico + parámetros seleccionados

```
Call:
 randomForest(formula = label ~ ., data = train, ntree = 20, do.trace = 1)
               Type of random forest: classification
                     Number of trees: 20
No. of variables tried at each split: 28

        OOB estimate of  error rate: 7.35%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 4017    3   17    6   11   20   25    6   24    3  0.02783156
1    1 4578   29   13    8    8   10   11   18    8  0.02263023
2   27   22 3866   53   26   20   32   57   57   16  0.07423372
3   16   18   98 3922   11  107   10   45   80   43  0.09839080
4   15   15   18    3 3797   15   28   14   28  139  0.06753438
5   39   11   23  155   21 3388   47   16   50   45  0.10724638
6   43   13   27   10   30   43 3931    4   33    3  0.04979454
7    8   29   68   28   37    7    2 4100   20  101  0.06818182
8   30   47   76   93   39   89   47   18 3564   60  0.12281565
9   24   11   25   59  119   54   17   84   46 3749  0.10482330
> |
```

Imagen 3: Matriz de confusión del modelo basado en *Random Forest*

Se observa que antes de subir a *Kaggle*, el modelo tiene un porcentaje de acierto en clasificaciones del 92,75% frente al 95,428% que se obtuvo en *Kaggle*.

Imagen 4: resultado del modelo basado en *Random Forest* en *Kaggle*

Las clasificaciones que más fallan es la clasificación entre el digito 4 y 9 y la clasificación entre el 3 y el 5.

V. Procesamiento de datos y resultados

Con el fin de intentar llevar a cabo una mejora del porcentaje de buenas clasificaciones del algoritmo *Random Forest,* se llevaron a cabo tres aproximaciones de procesado de los datos.

La primera aproximación consistió en eliminar del conjunto de datos aquellas columnas en las que todos los dígitos contuvieran el valor cero, ya que todas las imágenes contienen un mismo valor en una misma columna está columna no aporta ninguna información relevante.

El código desarrollado para este procesamiento fue el siguiente [3]:

```
#remove zeros
train_orig <- train
test_orig <- test
library(caret)
nzv.data <- nearZeroVar(train, saveMetrics = TRUE)
drop.cols <- rownames(nzv.data)[nzv.data$nzv == TRUE]
train <- train[,!names(train) %in% drop.cols]
test <- test[,!names(test) %in% drop.cols]
```

Tras la eliminación de las columnas de ceros, se procedió a implementar un modelo a través del algoritmo *Random Forest*. Los resultados de este primer procesamiento son los siguientes:

```
Call:
 randomForest(formula = label ~ ., data = train, ntree = 20, do.trace = 1)
               Type of random forest: classification
                     Number of trees: 20
No. of variables tried at each split: 15

        OOB estimate of  error rate: 7.32%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 3986    0   18   16    6   35   21    5   35   10  0.03533398
1    0 4571   26   24    7   10    7   15   17    7  0.02412468
2   40   21 3861   43   34   17   33   50   49   28  0.07543103
3   12   20  103 3899   12  131   17   38   81   38  0.10388416
4    9    7   26    4 3793   15   37   24   20  137  0.06851670
5   32   18   15  136   20 3436   52   13   47   26  0.09459816
6   40    9   40   11   31   53 3922    1   26    3  0.05174081
7    5   23   88   17   44    6    4 4119   19   76  0.06407635
8   18   34   68  102   44   72   39   22 3597   67  0.11469358
9   28   13   35   47  126   57   11   84   46 3741  0.10673352
> |
```

Imagen 5: Matriz de confusión del modelo *Random Forest* removiendo columnas de ceros.

Se observa que no existe una mejora en el porcentaje de acierto en clasificación. El porcentaje de buenas clasificaciones desciende un 0.02% respecto a la aplicación del modelo entrenado con los datos sin modificar.

En *Kaggle* se obtiene el siguiente resultado:

Imagen 6: resultado del modelo basado en *Random Forest* en *Kaggle*

El porcentaje de aciertos en clasificación en *Kaggle* también disminuye, por lo que esta primera aproximación quedó descartada.

En la segunda aproximación se llevó a cabo una modificación en la escala de grises de las imágenes del dataset. Como se observó en la imagen 1, el contorno de los dígitos del dataset no está bien definido. Para la mejora de la precisión de este contorno, se desarrolló el siguiente programa en el lenguaje de programación java, en el cual se descartaba el valor de aquellos pixeles que tuvieran un valor menor de 10.

El código desarrollado para esta aproximación es el siguiente:

grises.java

```java
import java.io.BufferedReader;

public class grises {
    public static void main(String [] args) {
        String csvFile = "bueno.csv";
        String line = "";
        String cvsSplitBy = ",";
        PrintWriter pw = null;
        try {
            pw = new PrintWriter(new File("grises out.csv"));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        StringBuilder sb = new StringBuilder();
        sb.append("label");
        sb.append(",");
        for(int j=1; j<=784; j++){
            sb.append("pixel"+(j-1));
            if(j<784) sb.append(",");
        }

        sb.append('\n');
        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            br.readline();
            while ((line = br.readLine()) != null) {
                String[] image = line.split(cvsSplitBy);

                sb.append(image[0]);sb.append(",");
                for (int i=1;i<image.length;i++) {
                    int aux=Integer.parseInt(image[i]);
                    if(aux<10){
                        sb.append(0);
                    }
                    else {
                        sb.append(aux);
                    }
                    sb.append(",");
                }
                sb.append('\n');
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        pw.write(sb.toString());
        pw.close();

    }
}
```

El conjunto de datos generado de este programa se utilizó posteriormente como conjunto de datos de entrenamiento de para el algoritmo *Random Forest*. El resultado de esta segunda aproximación fue el siguiente:

```
        OOB estimate of  error rate: 5.85%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 4021    0   13    6    9   19   29    1   28    6  0.02686350
1    0 4607   20   18    3    5    5   11   10    5  0.01643894
2   26   12 3911   42   32   18   25   47   53   11  0.06368207
3   19    7   99 3970    4   89   12   42   74   35  0.08756608
4   11   10   11    2 3851    9   31   12   25  110  0.05427308
5   28    7   17  101   20 3485   50    2   55   30  0.08168643
6   31    6   15    2   16   42 4003    3   18    1  0.03239062
7    4   25   59   18   31    4    2 4165   15   78  0.05362418
8   15   30   56   61   35   72   26   13 3703   52  0.08860448
9   24   11   24   45  114   40    3   59   41 3827  0.08619866
```

Imagen 7: Matriz de confusión del modelo *Random Forest* en escala de grises

Se observa que existe una mejora considerable de acierto en clasificación. El porcentaje de buenas clasificaciones asciende un 1.5% respecto a la aplicación del modelo entrenado con los datos sin modificar.

En *Kaggle* se obtiene el siguiente resultado*:*

Imagen 8: resultado del modelo basado en *Random Forest* en *Kaggle*

La tercera aproximación con la que se trabajó fue intentar reducir el tamaño de la imagen. La imagen original tiene una dimensión de 28x28 píxeles, lo que se traduce en 784 atributos. El objetivo de esta aproximación era reducir el numero de atributos para observar si de esta manera el porcentaje de buenas clasificaciones mejoraba.

Se redujo la imagen de 28x18 píxeles a una imagen de 14x14pixeles, reduciendo cuatro veces el número de atributos.

El algoritmo consiste en formar un cuadrado con cuatro pixeles de la imagen y proyectar esos cuatro puntos en un único punto resultando una nueva imagen. En una aproximación llevada a cabo por un alumno durante el curso anterior, el valor de la proyección de los cuatro puntos se llevó a cabo a través de la media de los cuatro valores de los píxeles. En la actual aproximación, se decidió llevar a cabo la reducción a través de la toma del valor máximo de los cuatro píxeles.

El origen de esta idea es que la media de los cuatro pixeles puede resultar un valor muy bajo y perder entonces información. Un ejemplo sería el siguiente: el trazado del número nueve, uno de los números que más problema tienen en la clasificación el algoritmo *Random Forest*. Podría trazarse el número nueve de tal manera que la parte superior del numero no quedara del todo cerrada y pudiera confundirse entonces.

La opción de asignar el máximo valor de los cuatro posibles podría reducir conflictos como el planteado anteriormente y reduciendo la cantidad de información eliminada.

El código desarrollado para esta aproximación es el siguiente:

ImageReduction.java

```
import java.io.BufferedReader;

public class ImageReduction{

public static int getMaxim(int a, int b, int c, int d) {
    int max=a;
    if(b>max) max=b;
    if(c>max) max=c;
    if(d>max) max=d;
    return max;
}


public static void generateTestDigitRecognizerTrain() throws FileNotFoundException{
    String csvFile = "train-kaggle.csv";
    String line = "";
    String cvsSplitBy = ",";
    PrintWriter pw;
    pw = new PrintWriter(new File("solution.csv"));
    StringBuilder sb = new StringBuilder();
    sb.append("label");sb.append(",");
    for(int j=0; j<196; j++){
        sb.append("pixel"+j);
        if(j<195) sb.append(",");
    }
    sb.append('\n');
    try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
        br.readLine();
        while ((line = br.readLine()) != null) {
            String[] image = line.split(cvsSplitBy);
            int contador = 0;
            sb.append(image[0]);sb.append(",");
            for(int k=1; k<730; k=k+56){
                for(int i=k; i<28+k; i=i+2){
                    if(contador!=0){
                        sb.append(",");
                    }
                    int value =
getMaxim(Integer.valueOf(image[i]),Integer.valueOf(image[i+1]),
                            Integer.valueOf(image[i+28]), Integer.valueOf(image[i+29]));

                    sb.append(value);
                    contador++;
                }
            }
            sb.append('\n');
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    pw.write(sb.toString());
    pw.close();
}

public static void generateTestDigitRecognizerTest() throws FileNotFoundException{
    String csvFile = "test.csv";
    String line = "";
    String cvsSplitBy = ",";
    PrintWriter pw;
    pw = new PrintWriter(new File("solution_test.csv"));
    StringBuilder sb = new StringBuilder();
```

ImageReduction.java

```
    for(int j=0; j<196; j++){
        sb.append("pixel"+j);
        if(j<195) sb.append(",");
    }
    sb.append('\n');
    try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
        br.readLine();
        while ((line = br.readLine()) != null) {
            String[] image = line.split(cvsSplitBy);
            int contador = 0;
            for(int k=0; k<730; k=k+56){
                for(int i=k; i<28+k; i=i+2){
                    if(contador!=0){
                        sb.append(",");
                    }
                    int value =
  getMaxim(Integer.valueOf(image[i]),Integer.valueOf(image[i+1]),
                            Integer.valueOf(image[i+28]), Integer.valueOf(image[i+29]));

                    sb.append(value);
                    contador++;
                }
            }
            sb.append('\n');
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    pw.write(sb.toString());
    pw.close();
}

public static void main (String []args) {
    try {
        generateTestDigitRecognizerTrain();
        generateTestDigitRecognizerTest();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
```

El conjunto de datos generado de este programa se utilizó posteriormente como conjunto de datos de entrenamiento de para el algoritmo *Random Forest*. El resultado de esta tercera aproximación fue el siguiente:

```
        OOB estimate of  error rate: 6.02%
Confusion matrix:
     0    1    2    3    4    5    6    7    8    9 class.error
0 4046    1   14    5   11   12   15    2   23    3  0.02081317
1    0 4583   24   11   12    7   11   10   21    5  0.02156277
2   21   18 3937   44   27   10   13   54   41   12  0.05745751
3   11   17   85 3971    7   99   12   32   88   29  0.08733624
4    9    8   13    4 3830    4   24   23   23  134  0.05943026
5   37    9   10  135   12 3457   49    6   53   27  0.08906456
6   26   16   10    1   19   47 3996    0   20    2  0.03408267
7    7   20   42   12   40    3    0 4148   18  111  0.05748693
8   22   40   39   74   41   67   28   14 3689   49  0.09205021
9   18   10   12   70  103   21    7   86   45 3816  0.08882521
```

Imagen 9: Matriz de confusión del modelo *Random Forest* en reducción de dimensión

Se observa que existe tampoco existe una mejora considerable de acierto en clasificación. El porcentaje de buenas clasificaciones desciende unas centésimas respecto a la aplicación del modelo entrenado con los datos sin modificar.

En *Kaggle* se obtiene el siguiente resultado*:*

Imagen 10: resultado del modelo basado en *Random Forest* en *Kaggle*

VI. Conclusiones

En los procesos de descubrimiento, el proceso de data mining a través del cual se aplican algoritmos de aprendizaje y se entrenan los modelos desarrollados de ellos, no es el paso más importante. Aunque la selección de un tipo de algoritmo frente a otro puede ayudar considerablemente a obtener mejores resultados, la etapa de comprensión de los datos y la etapa de preprocesamiento de los datos son las etapas más críticas.

La etapa de comprensión de los datos nos ayuda a tener un conocimiento un poco más profundo sobre el dominio del problema que queremos resolver. Sin un buen conocimiento del dominio podemos llevar a cabo un mal preprocesamiento de los datos y llevarnos a eliminar atributos de los datos que podrían ser contener información importante de los mismos.

Sobre el estudio de los diferentes algoritmos de clasificación aplicados, el algoritmo que mejores resultados a obtenido es el algoritmo *Random Forest*, aunque también se observa un porcentaje de clasificación satisfactorio mediante el algoritmo *SVM*. El presente trabajo se ha llevado a cabo a través de diferentes aproximaciones con el algoritmo *Random Forest* como base, descartando el uso de *SVM*. Podría plantearse el uso del *SVM* para trabajos futuros.

Aunque las aproximaciones utilizadas en el preprocesamiento de los datos en este trabajo no hayan sido demasiado satisfactorias en cuanto a porcentajes altos de clasificación, se puede decir que ha sido satisfactorio desde el punto de vista sobre que aproximaciones tomar o no, por lo que podría continuarse el estudio del problema planteado buscando nuevas alternativas a las estudiadas en presente trabajo.

El uso de la plataforma *Kaggle* para el desarrollo de este proyecto ha dado la oportunidad de conocer nuevas alternativas para continuar el aprendizaje en esta área de la informática. La plataforma ofrece a los usuarios diferentes guías sobre cómo participar en las competiciones y permite la creación de canales de comunicación entre los diferentes usuarios, donde cada participante puede abrir temas de debate y ayudar al resto de participantes a obtener mejores puntuaciones.

El único inconveniente de la plataforma es que tiene limitado el número de participaciones por usuario en un mismo día, por lo que no te permite hacer demasiadas comprobaciones reales en la competición sobre tu modelo

VII. Referencias

[1] https://en.wikipedia.org/wiki/MNIST_database

Última consulta: 12 de enero de 2019.

[2] https://www.kaggle.com/c/digit-recognizer

Última consulta: 12 de enero de 2019.

[3] https://api.rpubs.com/dardodel/digit_recognition_KNN_NN_SVM

Última consulta: 12 de enero de 2019.

[4] LeCun, Y. (1998). The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Última consulta: 12 de enero de 2019.

[5] R Documentation (2019) Package 'naivebayes'. https://cran.rproject.org/web/packages/naivebayes/naivebayes.pdf

Última consulta: 12 de enero de 2019.

[6] https://rischanlab.github.io/SVM.html

Última consulta: 12 de enero de 2019.

[7] Artificial Neural Networks as Models of Neural Information Processing | Frontiers Research Topic.https://www.frontiersin.org/research-topics/4817/artificial-neural-networks-as-models-of-neural-information-processing «

Última consulta: 12 de enero de 2019.