Máster Universitario en

Ingeniería **C**omputacional y **S**istemas **I**nteligentes

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Konputazio Zientziak eta Adimen Artifiziala Saila –
Departamento de Ciencias de la Computación e Inteligencia Artificial

K I S A

I C S I

Tesis de Máster

Análisis predictivo para clasificar dígitos escritos a mano utilizando la base de datos MNIST

Omar Alexander Ruiz-Vivanco

Tutor(a/es)

Iñaki Inza
Departamento de Ciencia de la Computación e Inteligencia Artificial
Facultad de Informática

informatika fakultatea
facultad de informática

KZAA /CCIA

Septiembre 2016

Resumen

En el presente trabajo de fin de máster se realiza una investigación sobre las técnicas de preproceso del dataset de entrenamiento y la aplicación de un modelo de predicción que realice una clasificación de dígitos escritos a mano. El conjunto de dataset de train y test son proporcionado en la competencia de Kaggle: Digit Recognizer y provienen de la base de datos de dígitos manuscritos MNIST.

Por tratarse de imágenes las técnicas de preproceso se concentran en obtener una imagen lo más nítida posible y la reducción de tamaño de la misma, objetivos que se logran con técnicas de umbralización por el método de Otsu, transformada de Wavelet de Haar y el análisis de sus componentes principales. Se utiliza Deep Learning como modelo predictivo por ajustarse a este tipo de datos, se emplean además librerías de código abierto implementadas en el lenguaje estádisto R.

Por último se obtiene una predicción con las técnicas y herramientas mencionadas para ser evaluada en la competencia de Kaggle, midiendo y comparando los resultados obtenidos con el resto de participantes.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este trabajo se fundamenta en la competicíon de Kaggle denominada: Digit Recognizer, cuya finalidad es el reconocimiento de dígitos escritos a mano. Kaggle es una plataforma en donde las empresas y los investigadores publican datos y estadísticas con el objetivo de brindar a participantes de cualquier parte del mundo la posibilidad de producir los mejores modelos predictivos a través de competiciones en donde se presentan un sinnúmero de estrategias para el mismo problema, los dataset para esta competición en particular son proporcionados por MNIST[1] que es una gran base de datos de dígitos escritos a mano de acceso público y usada ampliamente para entrenamiento y pruebas en el campo de aprendizaje automático.

1.1. Problemática

Las utilidades que se obtienen hoy en día del reconocimiento automático de dígitos a partir de una imágen pueden ser muy variadas, lo importante es desarrollar una técnica eficiente para deducir una función a partir de datos de entrenamiento, lo que constituye la esencia del aprendizaje supervisado. "Enseñar", a un computador a reconocer dígitos escritos a mano por distintas personas en donde factores como intensidad del trazo, curvatura, posición de la mano, etc., generan múltiples variaciones del dibujo final, constituye el problema a resolver.

[1]Modified National Institute of Standards and Technology.

1.1.1. Pregunta

¿Cómo se obtienen scores similares a los publicados por los competidores de Kaggle para clasificar dígitos escritos a mano utilizando la base de datos MNIST?

1.1.2. Hipótesis

Utilizando técnicas de preproceso de variables y tratamiento de imágenes se puede construir modelos predictivos que lleguen a scores similares a los publicados por los competidores de Kaggle para clasificar dígitos escritos a mano utilizando la base de datos MNIST.

1.2. Objetivos

1.2.1. Objetivo General

Identificar el dígito de una imagen que contiene un solo número escrito a mano de la base de datos MNIST.

1.2.2. Objetivos Específicos

- Utilizar técnicas de preproceso de variables que faciliten el análisis predictivo y optimicen el tiempo de clasificación.
- Construir un modelo predictivo eficaz para clasificar dígitos escritos a mano.
- Analizar los resultados obtenidos y compararlos con los participantes de la competencia de Kaggle: Digit Recognizer
- Calcular la efectividad de predicción (accuracy) del modelo obtenido.

1.3. Conjunto de datos

1.3.1. Base de datos: MNIST

La base de datos MNIST está compuesta por 60.000 ejemplos de entrenamiento y 10.000 ejemplos de prueba, esta partición es estándar en LeCun et al. [16], tomada como base en la mayoría de trabajos que utilizan este dataset, se construyó a partir de NIST[2] Special Database 3 (SD-3) y Special Database 1 (SD-1), que son bases de datos que contienen las imágenes binarias de dígitos escritos a mano, SD-3 como conjunto de entrenamiento y SD-1 como conjunto de prueba. SD-1 se elaboró con la colaboración de los estudiantes de la escuela secundaria y contiene 58,527 imágenes de dígitos escritos por 500 autores diferentes, SD-3 se recogió entre los empleados de la oficina del censo, y es mucho más limpia y más fácil de reconocer que SD-1, donde los bloques de datos de cada escritor aparecieron en secuencia.

Sacar conclusiones razonables a partir de experimentos de aprendizaje requiere que el resultado sea independiente de la elección del conjunto de entrenamiento y de prueba entre el conjunto completo de muestras, por lo que, fueron mezclados los conjuntos de datos del NIST para construir una nueva base de datos, SD-1 se dividió en dos, los caracteres escritos por los primeros 250 escritores entraron en el nuevo conjunto de entrenamiento, los 250 restantes se colocaron en el conjunto de prueba, formándose dos conjuntos con cerca de 30.000 ejemplos cada uno, el nuevo conjunto de entrenamiento se completó con suficientes ejemplos de SD-3, para completar 60.000 patrones de entrenamiento, del mismo modo el nuevo equipo de prueba se completó con SD-3 formando 60.000 patrones de prueba.

En el conjunto de entrenamiento MNIST se tomó 30.000 patrones de SD-3 y 30.000 patrones de SD-1, para el conjunto de prueba se tomó 5.000 patrones de SD-3 y 5.000 patrones de SD-1, según se encuentran disponibles publicamente en los archivos[3] en donde los dataset están almacenados en un formato diseñado para el almacenamiento de vectores y matrices multidimensionales, utilizando código escrito en R [21] se realizó la tarea de leer los 4 archivos siguientes:

- train-images-idx3-ubyte: conjunto de entrenamiento (imágenes)

[2]National Institute of Standards and Technology.
[3]http://yann.lecun.com/exdb/mnist/index.html

- train-labels-idx1-ubyte: conjunto de entrenamiento (etiquetas)
- t10k-images-idx3-ubyte: conjunto de prueba (imágenes)
- t10k-labels-idx1-ubyte: conjunto de prueba (etiquetas)

Los archivos de imágen tienen una dimensión de 784 columnas con valores entre 0 para el color blanco y 255 para el color negro y representan los píxeles que conforman una imágen de 28x28 en escala de grises. Luego del proceso de lectura de los archivos, en "train_MNIST", y en "test_MNIST"se almacena un listado en donde sus elementos son: "n"para el total de valores del dataset, "x"para el conjunto de 784 variables, y "y"para el valor del dígito de la imágen.

La función "ver_digito"proporciona la visualización del dígito.

```
ver_digito <- function(arr784, col=gray(12:1/12), ...) {
image(matrix(arr784, nrow=28)[,28:1], col=col, ...)
}
```

Descripción de train_MNIST con 60.000 observaciones y 785 columnas.

```
summary(train_MNIST)

##   Length   Class  Mode
## n        1 -none- numeric
## x 47040000 -none- numeric
## y    60000 -none- numeric

str(train_MNIST)

## List of 3
##  $ n: int 60000
##  $ x: int [1:60000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##  $ y: int [1:60000] 5 0 4 1 9 2 1 3 1 4 ...
```

Dígito e imágen de la posición 48797:

```
train_MNIST$y[48797]

## [1] 8

ver_digito(train_MNIST$x[48797,])
```

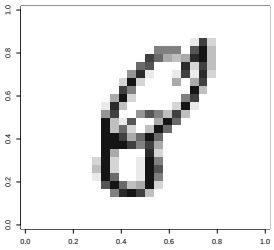


Figura 1.1: Dígito de train_MNIST posición #48797

Descripción de test_MNIST con 10.000 observaciones y 785 columnas.

```
summary(test_MNIST)

##   Length  Class  Mode
## n       1 -none- numeric
## x 7840000 -none- numeric
## y   10000 -none- numeric

str(test_MNIST)

## List of 3
##  $ n: int 10000
##  $ x: int [1:10000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
##  $ y: int [1:10000] 7 2 1 0 4 1 4 9 5 9 ...
```

Dígito e imágen de la posición 3698:

```
test_MNIST$y[3698]

## [1] 6

ver_digito(test_MNIST$x[3698,])
```

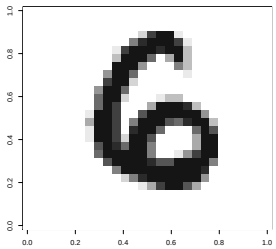


Figura 1.2: Dígito de train_MNIST posición #3698

1.3.2. Dataset de Kaggle: Digit Recognizer

En la competencia de Kaggle: Digit Recognizer [10] se propone la utilización de la base de datos MNIST, pero con una partición diferente, los 70.000 elementos de entrenamiento y prueba se han mezclado y dividido en los archivos "train.cvs", con 42.000 elementos y "test.cvs", con 28.000 elementos. En los objetos "train_kaggle_m", y "test_kaggle_m", se importan los datos de los archivos de Kaggle, haciendo notar que sólo en el conjunto de entrenamiento se tiene la variable clase denominada "label".

Descripción de train_kaggle_m con 42.000 observaciones y 785 columnas.

```
dim(train_kaggle_m)

## [1] 42000   785

train_kaggle_m[1,1:6]

##  label pixel0 pixel1 pixel2 pixel3 pixel4
##      1      0      0      0      0      0
```

Descripción de test_kaggle_m con 28.000 observaciones y 784 columnas.

```
dim(test_kaggle)

## [1] 28000   784

test_kaggle_m[1,1:6]

## pixel0 pixel1 pixel2 pixel3 pixel4 pixel5
##      0      0      0      0      0      0
```

El elemento #1 de train_MNIST corresponde al elemento #9262 de test_Kaggle_m, mientras que el elemento #2 de train_MNIST corresponde al elemento #8462 de test_Kaggle_m.

```
par(mfcol=c(2,2))
ver_digito(train_MNIST$x[1,])
ver_digito(train_MNIST$x[2,])
ver_digito(test_kaggle_m[9262,])
ver_digito(test_kaggle_m[8462,])
par(mfcol=c(1,1))
```

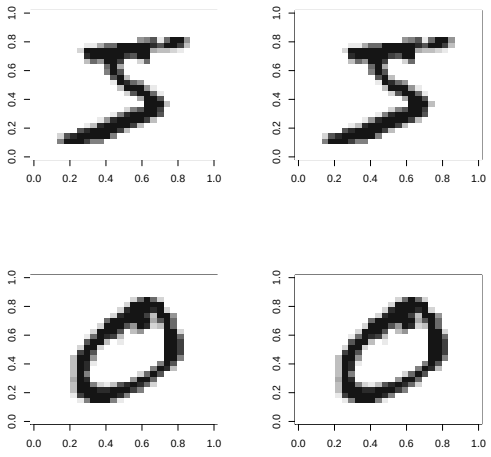


Figura 1.3: Dígitos de train_MNIST (izq.), dígitos de test_kaggle_m (der.)

El elemento #1 de train_Kaggle_m corresponde al elemento #53940 de train_MNIST, mientras que el elemento #2 de train_Kaggle_m corresponde al elemento #1927 de train_MNIST.

```
par(mfcol=c(2,2))
ver_digito(train_kaggle_m[1,-1])
ver_digito(train_kaggle_m[2,-1])
ver_digito(train_MNIST$x[53940,])
ver_digito(train_MNIST$x[1927,])
par(mfcol=c(1,1))
```

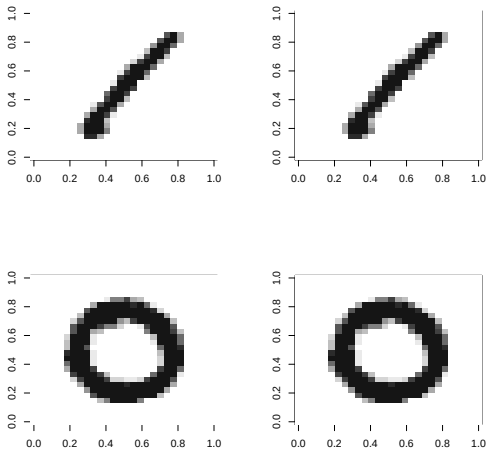


Figura 1.4: Dígitos de train_kaggle_m (izq.), dígitos de train_MNIST (der.)

La Figura 1.5 representa los 100 primeros números del dataset y su etiqueta.

```
par(mfrow = c(10,10), mai = c(0,0,0,0))
for(i in 1:100) {
ver_digito(train_kaggle_m[i,-1], axes = FALSE)
text( 0.2, 0, train_kaggle_m[i,1], cex = 3, col = 2, pos = c(3,4))
}
par(mfrow=c(1,1))
```

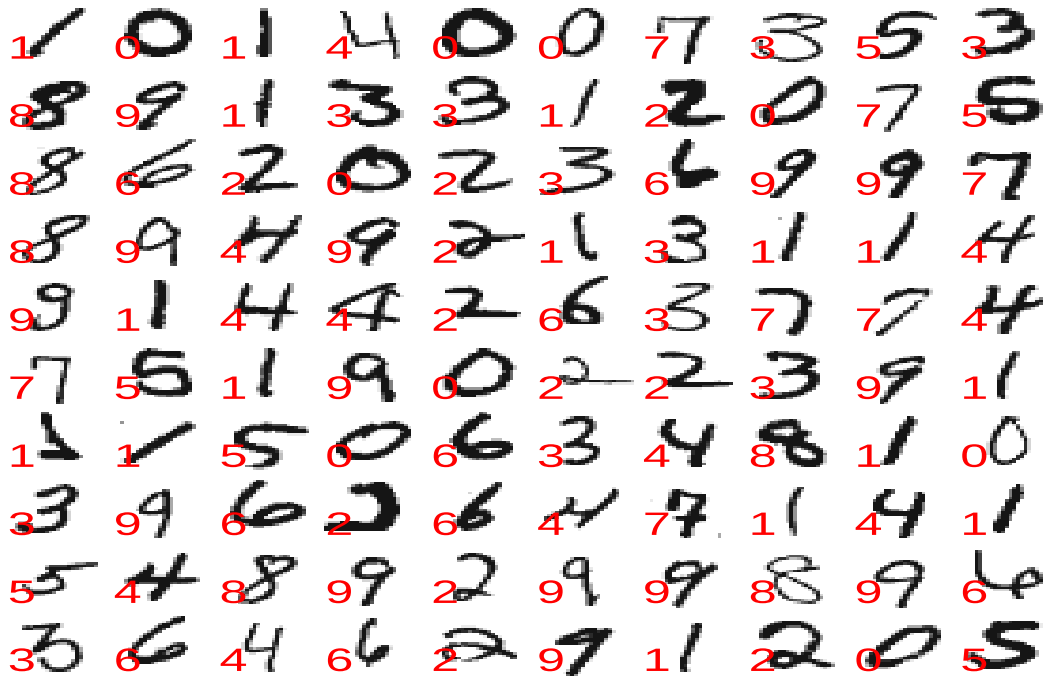


Figura 1.5: Pimeros 100 dígitos con su etiqueta.

Capítulo 2

Estado del arte

En esta sección se realiza un acercamiento formal hacia las producciones intelectuales que existen sobre trabajos realizados con anterioridad en la base de dígitos manuscritos MNIST, los diferentes preprocesos y los métodos de clasificación utilizados en cada una de las experimentaciones previas. Conocer otras investigaciones permitirá aclarar ideas, con la intención de definir, delimitar, y enfocar la investigación iniciando un camino propio. Existe diversidad de fuentes en donde se puede encontrar referencias al dataset de MNIST, con la finalidad de exponer las más relevantes se incluyen algunas como publicaciones científicas de autores destacados en el tema en los últimos años, los reportes en blogs especializados y las aportaciones de los competidores de Kaggle.

Yann LeCun[1] es quizá uno de los referentes en Deep Learning un modelo con buenos resultados sobre la base MNIST, en [17] una de sus publicaciones más representativas realiza un detallado análisis sobre redes convolucionales, Figura 2.1, redes para transformar gráficos[2] [3], Figura 2.2, y métodos de entrenamiento discriminativo[3] para la secuencia de etiquetado.

[1] Yann LeCun es Director de AI Research en Facebook, y profesor en New York University.

[2] Graph transformers networks: muestra cómo un gráfico puede ser usado para representar hipótesis de segmentación de una imagen que representa una secuencia de dígitos

[3] Modelan la dependencia de una variable no observada y sobre una variable observada x dentro de un marco probabilista.

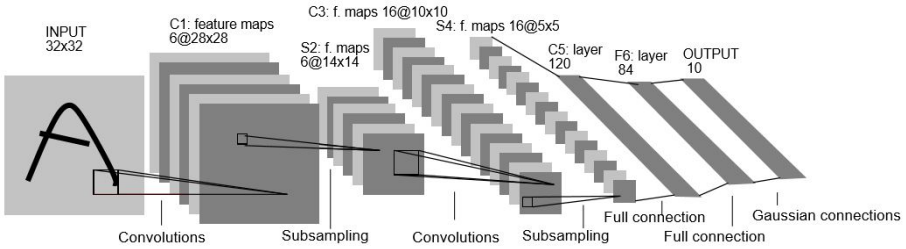


Figura 2.1: Red convolucional (LeNet5), cada plano es un mapa de características. Fuente: LeCun et al. [17]

Las redes convolucionales [18] mantienen el concepto de capas, con la diferencia de que cada neurona de una capa recibe sólo algunas de las conexiones entrantes de las neuronas de la capa anterior. Esto para especializar a una neurona en una región de la lista de números de la capa anterior, reduciendo drásticamente el número de pesos y de multiplicaciones innecesarias. En esta publicación se muestra la construcción de sistemas que integran la segmentación, extracción de características, clasificación, y construción de modelos de aprendizaje aplicables al reconocimiento de escritura y de rostros.

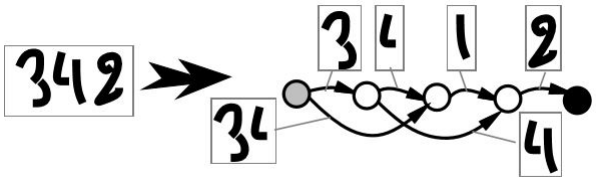


Figura 2.2: Graph transformers networks. Fuente: Bottou et al. [3]

El Cuadro 2.1 muestra algunos de los resultados obtenidos en la publicación, la experimentación se realizó con los dataset de la base MNIST, el dataset train de 60.000 registros y el dataset test de 10.000, la primera columna corresponde a las características del clasificador utilizado, la segunda detalla el preproceso aplicado al dataset y la tercera es el error de predicción.

Clasificador	Preproceso	Tasa de error
SVMs		
SVM deg 4 polynomial	enderezamiento	1,1
Reduced Set SVM deg 5 polynomial	enderezamiento	1,0
Virtual SVM deg-9 poly [distortions]	ninguno	0,8
Redes Neuronales		
2-layer, 300 hidden	enderezamiento	1,6
2-layer, 1000 hidden	ninguno	4,5
2-layer, 1000 hidden, [distortions]	ninguno	3,8
3-layer, 300+100 hidden	ninguno	3,05
3-layer, 300+100 hidden [distortions]	ninguno	2,5
3-layer, 500+150 hidden	ninguno	2,95
3-layer, 500+150 hidden [distortions]	ninguno	2,45
Redes Convolucionales		
LeNet-1	reducción 16x16	1,7
LeNet-4	ninguno	1,1
LeNet-5	ninguno	0,95
LeNet-5	gran deformación	0,85
LeNet-5	deformación	0,8
Boosted LeNet-4	deformación	0,7

Cuadro 2.1: Resultados de experimentación sobre MNIST en LeCun et al. [17]

En Simard et al. [23] la práctica más importante es conseguir un conjunto de entrenamiento lo más grande posible, esto con la adición de datos distorsionados. Se utilizan las redes neuronales convolucionales, consideradas las más adecuadas para muchos problemas de reconocimiento visual. La estructura propuesta para obtener los resultados deseados no requiere de complejos métodos. Para el caso del reconocimiento de la escritura, se menciona que las deformaciones elásticas corresponden a oscilaciones incontroladas de los músculos de la mano, amortiguadas por la inercia. Las distorsiones utilizadas son simples rotaciones logradas por desplazamiento de las imágenes. Esto se hace mediante el cálculo para cada píxel de una nueva ubicación de destino con respecto a la ubicación original. En la experimentación se utilizan dos

modelos las redes neuronales y las redes convolucionales.

El Cuadro 2.2 muestra los resultados con los modelos mencionados utilizando únicamante el dataset train de MNIST particionado para obtener un nuevo dataset train con los primeros 50.000 registros y el nuevo datset test con los 10.000 registros siguientes.

Clasificador	Preproceso	Tasa de error
Redes Neuronales		
2-layer, 800 hidden, Cross-Entropy Loss	none	1,6
2-layer, 800 hidden, cross-entropy	deformación afín	1,1
2-layer, 800 hidden, MSE	deformación elástica	0,9
2-layer, 800 hidden, cross-entropy	deformación elástica	0,7
Redes Convolucionales		
Convolutional net, cross-entropy	deformación afín	0,6
Convolutional net, cross-entropy	deformación elástica	0,4

Cuadro 2.2: Resultados de experimentación sobre MNIST en Simard et al. [23]

En Cireşan et al. [7] se parte de lo mencionado en [23], enfatiza la utilización de muchas capas ocultas con bastantes neuronas por capa, un preproceso de deformación de imágenes, y la utilización de tarjetas gráficas que aceleran enormemente el proceso de aprendizaje. La deformación de imágenes da como resultado el aumento de muestras de entrenamiento permitiendo configurar redes neuronales con muchos pesos, esto crea insensibilidad a la variabilidad en su clase. Se combina rotación, escalado e inclinación horizontal en este preproceso, según parámetros como:

- σ y α: para emular las distorsiones elásticas provocadas por oscilaciones incontroladas de los músculos de la mano.
- β: un ángulo al azar de [-β,+β] que describe la rotación o el corte horizontal. Para el corte la $tan\beta$ define la relación entre el desplazamiento horizontal y altura de la imagen.
- γ_x, γ_y: para la escala horizontal y vertical, seleccionados al azar de [1-γ/100, 1+γ/100]

Los parámetros para la experimentación son: $\sigma = 5,0 - 6,0$, $\alpha = 36,0 - 38,0$, $\gamma = 15 - 20$ y $\beta = 7,5°$ para dígitos 1 y 7, $\beta = 15,0°$ para el resto de dígitos. El Cuadro 2.3 muestra los mejores resultados de esta publicación, realizada con el dataset MNIST con 60.000 registros para train y 10.000 para test.

Clasificador	Preproceso	Tasa de error
3-layer, 1000+500 hidden	deformación elástica	0,49
4-layer, 1500+1000+500 hidden	deformación e.	0,46
5-layer, 2000+1500+1000+500 hidden	deformación e.	0,41
6-layer, 2500+2000+1500+1000+500 hidden	deformación e.	0,35

Cuadro 2.3: Resultados de experimentación sobre MNIST en Cireşan et al. [7]

En [5] Arnol Candel[4] realiza un análisis a detalle del modelo Deep Learning utilizado en la experimentación, mediante la utilización de la función h2o.deeplearning, estos resultados se obtienen con el dataset de MNIST originales sin ningún tipo de tratamiento previo, es decir sin utilizar capas convolucionales o aumento de datos, logrando un error de predicción de 0,0083 que se convierte en el mejor resultado conseguido hasta ahora para este dataset, esto en un tiempo de procesamiento de 8 horas en 10 nodos, Figura 2.3.

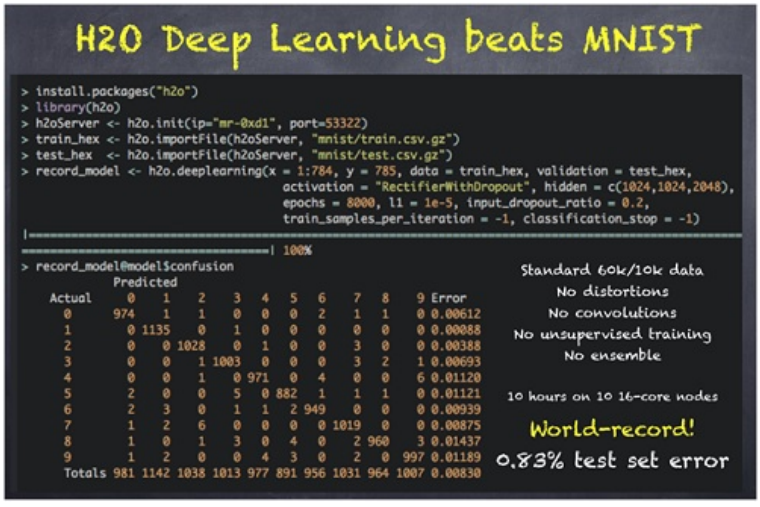


Figura 2.3: Configuración de h2o - world record. Fuente: KDnuggets. [15]

[4]Dr. Arno Candel, arquitecto jefe en H2O.ai.

En Kaggle [10] existe secciones en donde los participantes pueden opcionalmente realizar preguntas y compartir sus scripts con la solución o parte de ella, esta colaboración suele ser comentada y valorada por otros participantes, es así que se menciona a continuación las entradas más valoradas de la competencia.

En [13] el autor utiliza una particición de 40.000 registros para train y 2.000 para test, en un modelo Deep Learning con redes convolucionales codificado en Python mediante la herramienta TensorFlow™ [8], una biblioteca de software de código abierto desarrollada por Google para el cálculo numérico mediante diagramas de flujo de datos, pudiéndose utilizar más de una CPU o GPU en computadoras de escritorio, en servidores o en dispositivos móviles.

```
In [23]: # visualisation variables
train_accuracies = []
validation_accuracies = []
x_range = []

display_step=1

for i in range(TRAINING_ITERATIONS):

    #get new batch
    batch_xs, batch_ys = next_batch(BATCH_SIZE)

    # check progress on every 1st,2nd,...,10th,20th,...,100th... step
    if i%display_step == 0 or (i+1) == TRAINING_ITERATIONS:

        train_accuracy = accuracy.eval(feed_dict={x:batch_xs,
                                                  y_: batch_ys,
                                                  keep_prob: 1.0})
        if(VALIDATION_SIZE):
            validation_accuracy = accuracy.eval(feed_dict={ x: validation_images[0:BAT
CH_SIZE],
                                                            y_: validation_labels[0:BA
TCH_SIZE],
                                                            keep_prob: 1.0})

            print('training_accuracy / validation_accuracy => %.2f / %.2f for step %d'
%(train_accuracy, validation_accuracy, i))

            validation_accuracies.append(validation_accuracy)

        else:
             print('training_accuracy => %.4f for step %d'%(train_accuracy, i))
        train_accuracies.append(train_accuracy)
        x_range.append(i)

        # increase display_step
        if i%(display_step*10) == 0 and i:
            display_step *= 10
    # train on batch
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys, keep_prob: DROPOUT})
```

Figura 2.4: Código para el conjunto de entrenamiento en Python. Fuente: Kliavin [13]

En esta experimentación no se realiza un preproceso previo, sin embargo en cada convolución se obtienen versiones reducidas del dataset como entrada de la siguiente capa, la exactitud de predicción obtenida en esta implementación es de 98,45 % con

un computador Intel® Core™ i7 de 2.9 GHz. Este participante ha recibido 150 votos y 39 comentarios con su reporte actualizado 36 veces. La Figura 2.4 muestra un extracto del código utilizado en la experimentación.

En [11] el autor usa un modelo de red convolucional, no realiza un preproceso en sí sobre el dataset, más bien rediseña la estructura del dataset con la función "reshape", en Python para obtener una dimensión de (42.000,1), en donde cada fila contiene la matriz de 28x28 píxeles que forman el dígito. Este participante ha recibido 43 votos y 9 comentarios con su reporte actualizado 31 veces, el score en Kaggle es de 98,614 % Top 100, en un tiempo aproximado de 18.000 seg. en la fecha 2015/11/26. La Figura 2.5 muestra un extracto del código utilizado en la experimentación.

```
In [8]: def CNN(n_epochs):
            net1 = NeuralNet(
                layers=[
                ('input', layers.InputLayer),
                ('conv1', layers.Conv2DLayer),      #Convolutional layer.  Params defined below
                ('pool1', layers.MaxPool2DLayer),   # Like downsampling, for execution speed
                ('conv2', layers.Conv2DLayer),
                ('hidden3', layers.DenseLayer),
                ('output', layers.DenseLayer),
                ],

            input_shape=(None, 1, 28, 28),
            conv1_num_filters=7,
            conv1_filter_size=(3, 3),
            conv1_nonlinearity=lasagne.nonlinearities.rectify,

            pool1_pool_size=(2, 2),

            conv2_num_filters=12,
            conv2_filter_size=(2, 2),
            conv2_nonlinearity=lasagne.nonlinearities.rectify,

            hidden3_num_units=1000,
            output_num_units=10,
            output_nonlinearity=lasagne.nonlinearities.softmax,

            update_learning_rate=0.0001,
            update_momentum=0.9,

            max_epochs=n_epochs,
            verbose=1,
            )
            return net1

        cnn = CNN(15).fit(train,target) # train the CNN model for 15 epochs
```

Figura 2.5: Configuración red convolucional en Python. Fuente: Khitalishvili [11]

En [12] el mismo autor de [11] expone su experimentación utilizando la herramienta h2o en R. Utiliza una configuración para red neuronal de 2 capas, obtiene en sus pruebas 0,9604216 en 42.966 seg. de proceso. Este participante ha recibido 33 votos y 5 comentarios con su reporte actualizado 107 veces, el score en Kaggle es de 97,800 %. La Figura 2.6 muestra un extracto del código utilizado en la experimentación.

Si bien es cierto hay muchas entradas en los foros de la competencia, pero pocas ofrecen detalles que aclaren la experimentación realizada, siendo que no es requisito presentar un análisis de la solución para poder participar y obtener una calificación, las mencionadas son los más representativos con la mejor votación y con suficientes detalles de la implementación y sus resultados.

```
## set timer
s <- proc.time()

## train model
model =
  h2o.deeplearning(x = 2:785,  # column numbers for predictors
                   y = 1,   # column number for label
                   training_frame = train_h2o, # data in H2O format
                   activation = "RectifierWithDropout", # algorithm
                   input_dropout_ratio = 0.2, # % of inputs dropout
                   hidden_dropout_ratios = c(0.5,0.5), # % for nodes dropout
                   balance_classes = TRUE,
                   hidden = c(100,100), # two layers of 100 nodes
                   momentum_stable = 0.99,
                   nesterov_accelerated_gradient = T, # use it for speed
                   epochs = 15) # no. of epochs
```

Figura 2.6: Configuración de H2o en R. Fuente: Khitalishvili [12]

Capítulo 3

Material y métodos

3.1. Introdución

Como se había mencionado con anterioridad los dataset tanto para entrenamiento como para pruebas fueron tomados de la competencia Digit Recognizer de Kaggle. A partir de estos archivos la fase de experimentación se realiza totalmente con el lenguaje de programación R, herramienta altamente difundida en trabajos cientifícos por investigadores en campos como minería de datos, bioinformática, matemáticas financieras, etc.

En la minería de datos se extrae información de un conjunto de datos para transformarla en una estructura más comprensible y manejable. La técnica a utilizar es el aprendizaje supervisado en donde se deduce una función a partir de datos de entrenamiento para obtener una etiqueta de clase, es decir la clasificación de dígitos del 0 al 9. Esta tarea se lleva a cabo realizando un análisis de los datos iniciales para encontrar las técnicas de preproceso más adecuadas para luego aplicar el modelo predictivo que maximice el grado de acierto en la predicción de los dígitos.

3.2. Fase de análisis del dataset

Los dígitos del dataset contienen varias formas de escritura con diferentes trazos, curvatura, forma, etc., es decir diferencias muy particulares de cada escritor, con el objetivo de encontrar tendencias en la escritura de los dígitos se analiza el nivel de

intensidad de cada uno, añadiendo esta nueva característica al dataset mediante la obtención de la media.

```
#label como factor
train_kaggle$label <- as.factor(train_kaggle$label)
#intensidad como nueva característica
train_kaggle$intensidad <- apply(train_kaggle[,-1], 1, mean)
#Promedios por etiqueta
intensidad_digito <- aggregate(train_kaggle$intensidad,
by = list(train_kaggle$label), FUN = mean)
```

La Figura 3.1 muestra la intensidad en la escritura de cada dígito, siendo el "0", uno de los más intensos al contrario el "1", el menos intenso. Esta característica de intensidad de trazo puede ser útil al momento de realizar el análisis predictivo.

```
library(ggplot2)
int_digito_plot <- ggplot(data=intensidad_digito,
aes(x=Group.1, y = x)) +
geom_bar(stat="identity")
int_digito_plot + xlab("etiqueta dígito") +
ylab("promedio de intensidad")
```

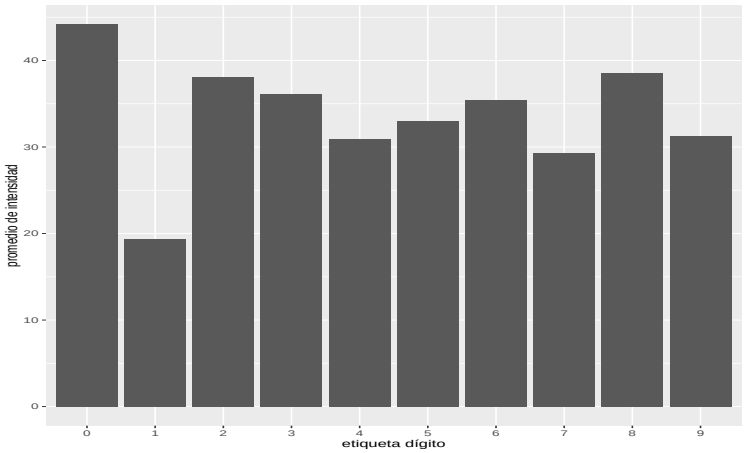


Figura 3.1: Histograma de intensidad por cada dígito

La escritura de los mismos dígitos difiere según el escritor tal es el caso del "4", cerrado en algunos casos, abierto en otros, el "7", con o sin una línea central, el "5", cerrándose en la parte inferior pareciéndose al "6", la Figura 3.2 corresponde a dígitos al azar y evidencia lo mencionado.

```
par(mfcol = c(3,10), mai=c(0,0,0,0))
for(i in 0:9) {
for(j in 1:3) {
id<-which(train_kaggle_m[,1]==i)[sample(rep(1:5),1)]
ver_digito(train_kaggle_m[id,-1], axes = FALSE)
}
}
par(mfcol=c(1,1))
```

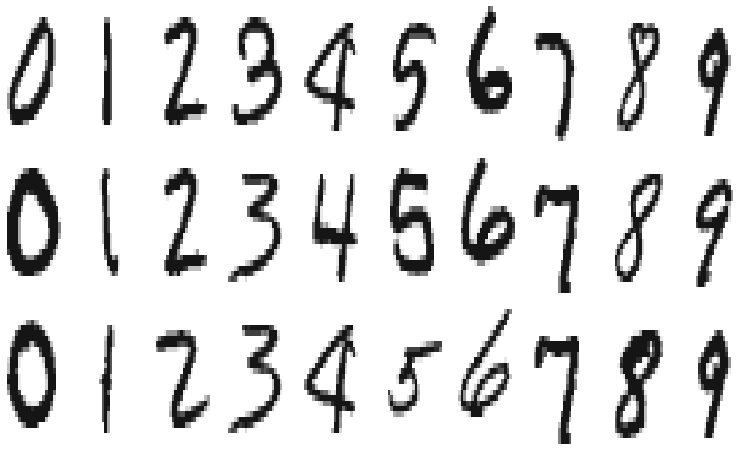


Figura 3.2: Dígitos al azar del dataset

El resultado de esta fase devuelve un dataset llamado **"train_kaggle_int"**, con la incorporación de una nueva característica denominada "intensidad".

```
dim(train_kaggle_i)

## [1] 42000   786
```

```
head(train_kaggle_i[1:5,781:786])

##   pixel779 pixel780 pixel781 pixel782 pixel783 intensidad
## 1        0        0        0        0        0   21.23597
## 2        0        0        0        0        0   56.89923
## 3        0        0        0        0        0   17.12372
## 4        0        0        0        0        0   19.16454
## 5        0        0        0        0        0   65.16964
```

3.3. Fase de preproceso de imágenes

Realizar un tratamiento previo en el dataset original se considera un paso obligado que se realiza antes de aplicar el modelo de predicción con el objetivo de obtener un insumo más limpio alcanzando un mejor resultado y mejorando el tiempo de proceso computacional, en algunos casos se necesita disminuir la dimensión de los datos, con la consecuencia de pérdida de información, en otros aclarar las imágenes con procesos de detección de píxeles más representativos. De la imagen tratada se extraen características importantes que aumentan el grado de acierto de las predicciones del modelo.

3.3.1. Reducción de la imagen

La reducción de la dimensión de los datos se logra aplicando la transformada de Wavelet Haar [26], "esta función transforma cualquier secuencia 3.1

$$(a_0, a_1, ..., a_{2n}, a_{2n+1}), \tag{3.1}$$

de cualquier longitud en una secuencia de dos componentes vectoriales 3.2

$$((a_0, a_1), ..., (a_{2n}, a_{2n+1})) \tag{3.2}$$

Si se multiplica por la derecha cada vector con la matriz H_2, se obtiene el resultado 3.3

$$((s_0, d_0), ..., (s_n, s_n)) \tag{3.3}$$

de una etapa de la transformada rápida de wavelet de Haar". La función devuelve las secuencias s y d y se utiliza la secuencia s para continuar con los cálculos. En el dataset train de Kaggle se utiliza dos etapas de cálculo equivalentes a tomar cuatro píxeles continuos y promediar su valor para obtener un nuevo píxel que remplace a los anteriores.

El objetivo es reducir la imágen de cada dígito manuscrito de 28x28 píxeles a una imágen de 14x14 píxeles. Para lo cuál se utiliza el principio de Wavelet de Haar. En R la función "ht", de la librería "binhf"[20], proporciona la transformada. El Algoritmo 1 es utilizado para lograr la reducción del dataset original de 784 columnas a 196.

Algoritmo 1 Transformada Wavelet de Haar

Entrada: Seleccionar matriz_original (28x28)

Salida: Matriz Reducida (14x14)

```
1: para i = 1 hasta 14 veces hacer
2:    para j = 1 hasta 14 veces hacer
3:       vector = matriz_original[i, j]
4:       haar = funcion_wavelet_haar(vector)
5:       haar = funcion_wavelet_haar(haar$s)
6:       matriz_reducida[i, j] = funcion_redondear(haar$s)
7:    fin para
8: fin para
9: devolver matriz_reducida
```

El resultado de esta fase devuelve un dataset con una reducción a 196 columnas más la columna de etiqueta, la Figura 3.3 muestra la reducción del dataset llamado: **"train_kaggle_r"**.

```
pos=4029
par(mfcol=c(1,2))
ver_digito(train_kaggle_m[pos,-1])
```

```
ver_digito_reduce(train_kaggle_r[pos,-1])
par(mfcol=c(1,1))
```

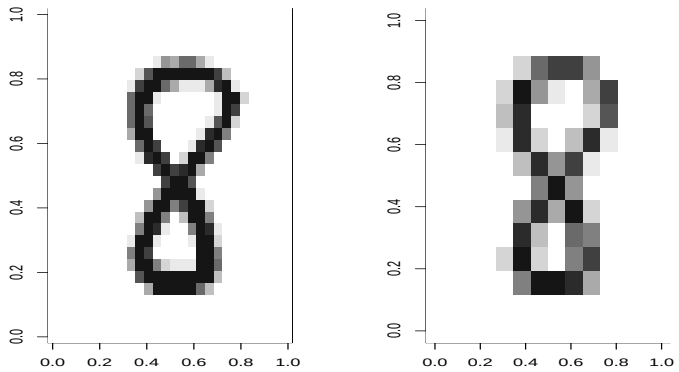


Figura 3.3: Dígito 28x28 (izq.), dígito 14x14 (der.)

Estructura interna del dataset "train_kaggle_r".

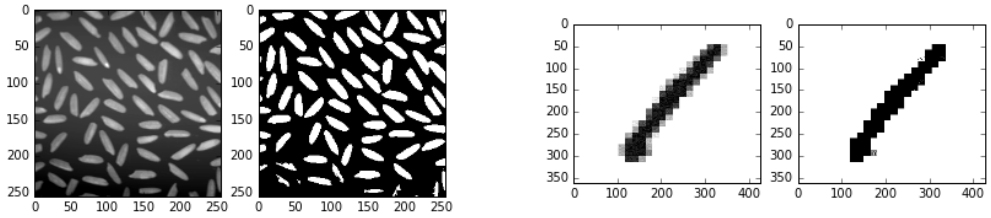
```
str(train_kaggle_r)

##  num [1:42000, 1:197] 1 0 1 4 0 0 7 3 5 3 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:42000] "1" "2" "3" "4" ...
##   ..$ : chr [1:197] "label" "pixel1" "pixel2" "pixel3" ...
```

3.3.2. Binarización de la imágen

La binarización es el método mediante el cuál se convierte los píxeles de una imagen de varios valores a únicamente dos (negro: 255 y blanco: 0). Para lo cual se establece un umbral sobre el cual todos los valores se convierten en 255 y por debajo de este en 0. El método de Otsu [25] encuentra el umbral óptimo (threshold) maximizando la varianza entre clases (between-class variance) mediante una búsqueda exhaustiva. La umbralización se utiliza cuando la diferencia entre la imagen y el

fondo existe una clara diferencia, Figura 3.4. Se fundamenta en la similitud entre los píxeles que forman el objeto principal y sus diferencias respecto al resto de objetos. La escena debería tener un fondo uniforme y objetos parecidos.



(a) Granos de arroz (b) 1er dígito dataset Kaggle

Figura 3.4: Ejemplos de umbralización por método Otsu.

A continuación se describe la fórmula de la técnica de Otsu basada en estadísticas sobre el histograma unidimensional de una imagen.

$$T = max(\sigma^2) \tag{3.4}$$

siendo

$$\sigma^2 = w_B(\mu_B - \mu)^2 + w_F(\mu_F - \mu)^2 \tag{3.5}$$

$$w_k = \sum_{i=0}^{k} p_i \tag{3.6}$$

$$\mu_k = \sum_{i=0}^{k} i.p_i \tag{3.7}$$

$$\mu_c = \frac{\mu_k}{\omega_k} \tag{3.8}$$

Donde p_i es la probabilidad de aparición de un determinado nivel i.

Las imágenes de la Figura 3.4 son obtenidas mediante la implementación del algoritmo descrito en [22]. Encontrar un umbral óptimo, es decir un número entre los valores de los píxeles que forman una imágen para utilizar como límite en la binarización se logra con el Algoritmo 2.

Algoritmo 2 Método de Otsu

Entrada: Seleccionar histograma de la imágen

Salida: Umbral óptimo

```
1: prob_acum = funcion_arreglo(256 elementos)
2: media_acum = funcion_arreglo(256 elementos)
3: prob_acum[1] = histograma[1]
4: long_hist = funcion_longitud(histograma)
5: para i = 2 hasta long_hist veces hacer
6:    prob_acum[i] = prob_acum[i-1] + histograma[i]
7:    media_acum[i] = media_acum[i-1] + (i-1) * histograma[i]
8: fin para
9: sigmaB2 = 0
10: inten_media = media_acum[long_hist-1]
11: sigmaB2max = 0
12: umbral_optimo = 0
13: para i = 1 hasta long_hist veces hacer
14:    clase1 = prob_acum[i]
15:    clase2 = 1 - clase1
16:    si (clase1 != 0) y (clase2 !=0) entonces
17:       m1 = media_acum[i] / clase1
18:       m2 = (inten_media - media_acum[i]) / clase2
19:       sigmaB2 = clase1 * funcion_cuadrado(m1 - inten_media) + clase2 * fun-
          cion_cuadrado(m2 - inten_media)
20:       si sigmaB2 es mayor que sigmaB2max entonces
21:          sigmaB2max = sigmaB2
22:          umbral_optimo = i - 1
23:       fin si
24:    fin si
25: fin para
26: devolver umbral_optimo
```

Con el umbral óptimo para cada imágen se cambian los píxeles con valores menores al límite a “0”, y los mayores a “1”, Algoritmo 3.

Algoritmo 3 Binarizar imágen

Entrada: Seleccionar imágen dígito

Salida: Matriz Binarizada

```
1: hist = funcion_histograma(imagen_digito)
2: otsu = funcion_umbral_optimo(hist)
3: x,y = funcion_dimension(imagen_digito)
4: para i = 1 hasta x veces hacer
5:    para j = 1 hasta y veces hacer
6:       si imagen_digito[i,j] es mayor que otsu entonces
7:          matriz_binarizada[i,j] = 1
8:       si no
9:          matriz_binarizada[i,j] = 0
10:      fin si
11:   fin para
12: fin para
13: devolver matriz_binarizada
```

Se aplica la binarización al dataset train original de 784 columnas, y al dataset train reducido de 196 columnas, además se aumenta la columna de intensidad a ambos para obtener un total de 4 dataset como resultado de esta fase: **"train_kaggle_bo"**, **"train_kaggle_br"**, **"train_kaggle_boi"**, y **"train_kaggle_bri"**. La Figura 3.5 muestra aleatoriamente dígitos de 0 a 9 con el resultado de aplicar la binarización.

```
par(mfrow = c(10,4), mai=c(0,0,0,0))
for(i in 0:9) {
id<-which(train_kaggle_m[,1]==i)[sample(rep(1:200),1)]
ver_digito(train_kaggle_m[id,-1], axes = FALSE)
ver_digito(train_kaggle_bo[id,-1], axes = FALSE)
ver_digito_reduce(train_kaggle_r[id,-1], axes = FALSE)
ver_digito_reduce(train_kaggle_br[id,-1], axes = FALSE)
}
par(mfrow=c(1,1))
```

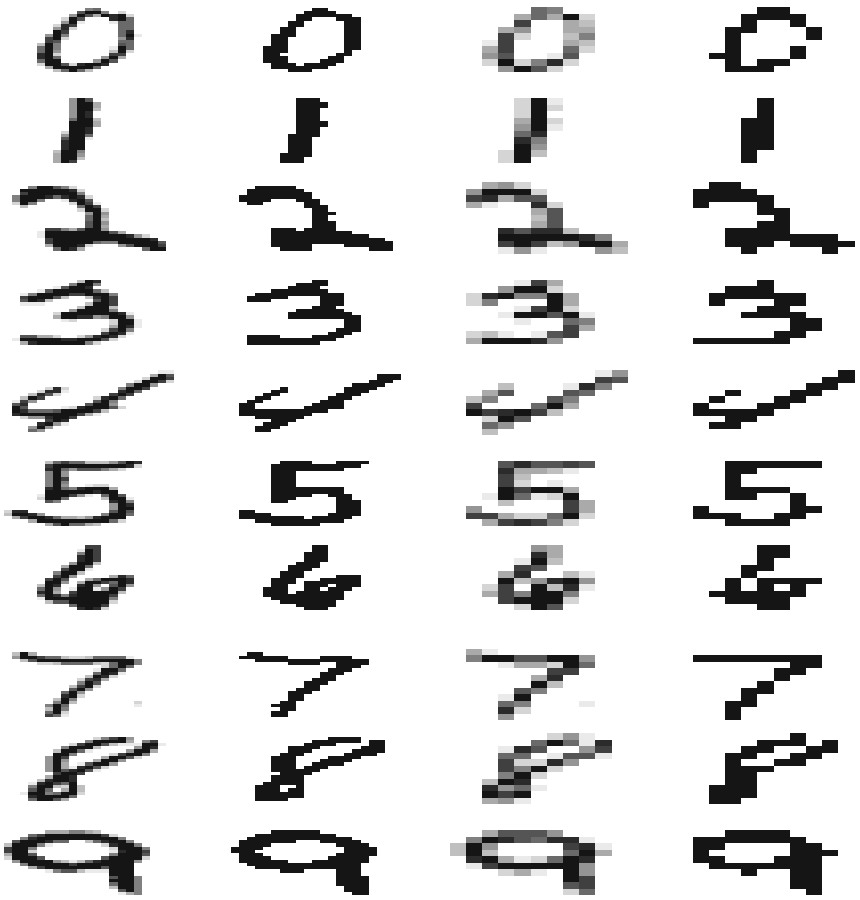


Figura 3.5: Imágen 28x28 (1ra.), imágen 28x28 binarizada (2da.), imágen 14x14 (3ra.), imágen 14x14 binarizada (4ta.)

3.4. Fase de extracción de nuevas variables

El análisis de componentes principales [27] es una técnica de transformación con el objetivo de identificar los principales factores que explican las diferencias en los datos y una descripción más comprimida de sus correlaciones. Constituye una herramienta para extraer tendencias generales de un conjunto de datos. PCA ordena los componentes principales por su significado ofreciendo una excelente base para la reducción de la dimensión de datos multidimensionales.

Este proceso identifica un grupo de variables ficticias que se forman de la com-

binación de las anteriores observadas. De esta forma se sintetizan los datos y se relacionan entre sí, sin recurrir a ninguna hipótesis previa sobre el significado de cada factor inicial.

Los componentes principales obtenidos mediante el proceso de cálculo de raíces y vectores característicos de una matriz simétrica contienen la mayoría de la varianza observada, evitando la información redundante. Para esto las variables deben ser incorreladas entre sí y se expresan como combinación lineal de las demás variables que han sido observadas. El porcentaje de mayor varianza mostrada en cada una de estas componentes se traduce en que las mismas contiene mayor cantidad de información.

En R la función "prcomp", devuelve los componentes mencionados accesados por el operador $, así:

- $sdev: es la raíz cuadrada autovalores.
- $rotation: son los autovectores.
- $x: nuevas coordenadas.

Se aplica PCA [6] a los archivos resultantes de la fase de preproceso:

- train_kaggle_bo
- train_kaggle_br

En la Figura 3.6 se observa que la varianza acumulada se va estabilizando en el valor de 1.00 según aumenta el número de elementos principales, demostrando que las primeras componentes contienen la mayor variabilidad de datos.

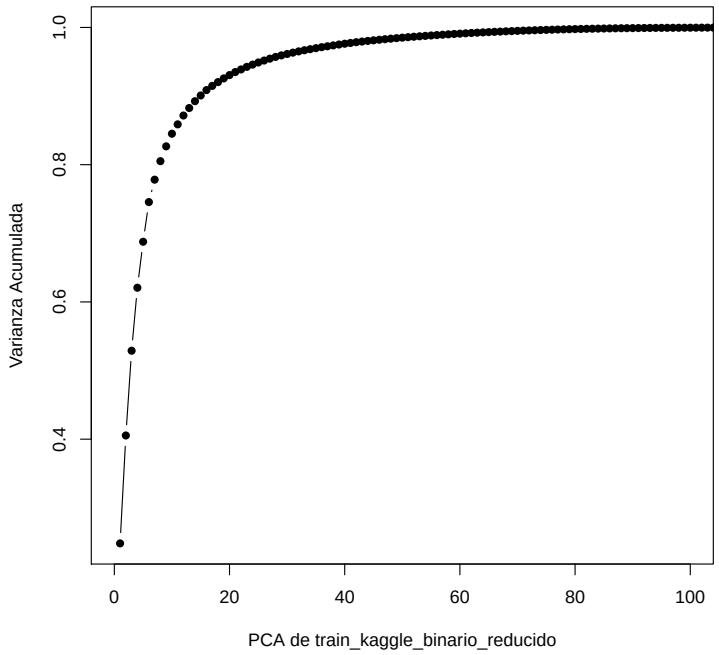


Figura 3.6: Relación entre la varianza y los componentes principales

Tabla de datos de las varianzas individual y acumulada por cada componente.

```
var_explicada_resumen <- var_explicada[seq(0,130,5),]
var_explicada_resumen
```

```
##     No_PCA Varianza_Individual Varianza_Acumulada
## 5        5        6.701483e-02          0.6877083
## 10      10        1.841086e-02          0.8451176
## 15      15        8.304337e-03          0.9008642
## 20      20        4.848596e-03          0.9305756
## 25      25        3.079354e-03          0.9489231
## 30      30        2.047122e-03          0.9613334
## 35      35        1.489238e-03          0.9698209
## 40      40        1.163358e-03          0.9762737
## 45      45        8.676169e-04          0.9812218
```

```
## 50      50        7.190846e-04          0.9851415
## 55      55        5.823073e-04          0.9882821
## 60      60        5.103256e-04          0.9909821
## 65      65        3.957617e-04          0.9932083
## 70      70        3.331187e-04          0.9950237
## 75      75        2.591817e-04          0.9964753
## 80      80        2.050236e-04          0.9976256
## 85      85        1.475926e-04          0.9984550
## 90      90        9.074207e-05          0.9990282
## 95      95        6.567375e-05          0.9994010
## 100    100        3.642688e-05          0.9996180
## 105    105        2.709439e-05          0.9997713
## 110    110        1.596963e-05          0.9998691
## 115    115        1.046909e-05          0.9999292
## 120    120        5.687470e-06          0.9999656
## 125    125        2.853417e-06          0.9999856
## 130    130        1.195858e-06          0.9999946
```

La Figura 3.7 identifica la importancia de los primeras componentes en la descripción del dataset train_kaggle_bo.

```
barplot(summary(pca_train_bo)$importance[2, ],
xlab = "PCA train binario original",
ylab = "Importancia",xlim = c(0,60), axes = TRUE)
```

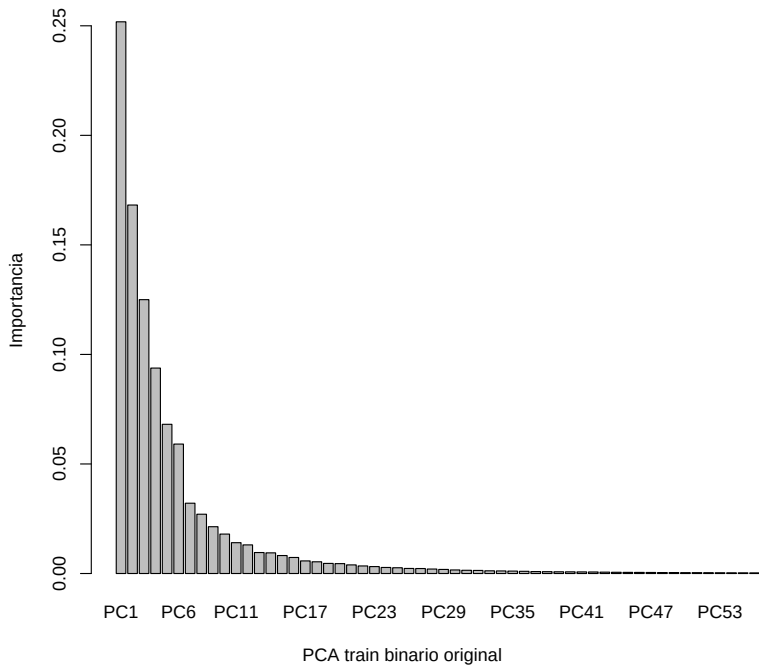


Figura 3.7: Componentes principales del dataset train binario original

Las primeras 64 componentes describen el 99.21 % de las caracteísticas del dataset: train_kaggle_bo.

```
sum(summary(pca_train_bo)$importance[2,][1:64])

## [1] 0.99209
```

La Figura 3.8 identifica la importancia de los primeras componentes en la descripción del dataset train_kaggle_br.

```
barplot(summary(pca_train_br)$importance[2, ],
xlab = "PCA train binario reducido",
ylab = "Importancia",xlim = c(0,60), axes = TRUE)
```

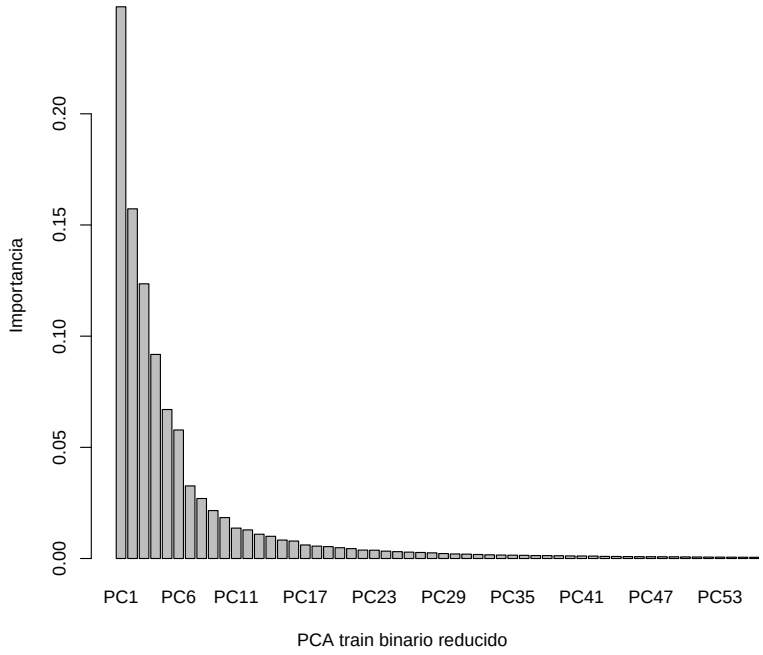


Figura 3.8: Componentes principales del dataset train binario reducido

Las primeras 64 componentes describen el 99.28 % de las caracteísticas del dataset: train_kaggle_br.

```
sum(summary(pca_train_br)$importance[2,][1:64])

## [1] 0.99281
```

Se aplica la reducción del dataset por PCA para las 64 primeras componentes, con lo que se obtiene dos nuevos dataset.

"train_kaggle_pca_bo", con 65 columnas, incluída la etiqueta del dígito.

```
str(train_kaggle_pca_bo)

##  num [1:42000, 1:65] 1 0 1 4 0 0 7 3 5 3 ...
##  - attr(*, "dimnames")=List of 2
```

```
##   ..$ : NULL
##   ..$ : chr [1:65] "label" "PC1" "PC2" "PC3" ...
```

"train_kaggle_pca_br", con 65 columnas, incluída la etiqueta del dígito.

```
str(train_kaggle_pca_br)

##  num [1:42000, 1:65] 1 0 1 4 0 0 7 3 5 3 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:65] "label" "PC1" "PC2" "PC3" ...
```

3.5. Fase de construcción del modelo predictivo

Esta fase tiene como objetivo encontrar los clasificadores que se ajusten a las caraterístcas del dataset de dígitos manuscritos. Deep learning es el método de aprendizaje automático que se utiliza en la construción del modelo.

3.5.1. Deep Learning

El deep learning [18] forma parte de un conjunto amplio de métodos de aprendizaje automático los cuales se basan en aprender representaciones de datos, mediante una serie de operaciones matemáticas sobre una lista de números, dando como resultado otra lista de números. Por ejemplo el reconocimiento de imágenes, se codifica como una lista de números, por tanto, la red neuronal recibiría tantos números en su entrada como píxeles tienen las imágenes o el caso de imágenes en color tres por cada píxel. Su salida bastaría con un solo número cercano a 1.0 para determinar si se trata de un rostro (figura, letra, dígito, etc.) o cercano a 0.0 si no reconoce un rostro, los valores intermedios se interpretan como inseguridad o probabilidad.

La arquitectura de las redes de neuronas [24] se basa en la conexión que tienen las neuronas en cada capa con la siguiente, estas conexiones tienen asociado un número llamado peso. La principal operación dentro de las redes consiste en multiplicar los valores de una neurona por los pesos de sus conexiones salientes, luego cada neurona

de la siguiente capa suma todos los input que recibe de las conexiones entrantes. En cada capa se utiliza una función de activación que se encarga de mantener los números producidos por cada neurona dentro de un rango razonable, números reales entre 0 y 1. La función más habitual es la sigmoide que es no lineal, es decir que si se grafican los valores de entrada y de salida el resultado no sería una línea.

Cuando se nombra a una red neuronal se lo hace asi: n-layer. La capa de salida se toma para representar los resultados de clase, 10 en el caso de los dígitos manuscritos de Kaggle. Se dimensiona la red neuronal por el número de neuronas o por el número de parámetros. La Figura 3.9 es una red neuronal de 2 capas, no se cuenta la capa de entrada, tiene 4 neuronas en su capa oculta más 2 neuronas en su capa de salida, total 6 neuronas. Tiene 20 pesos (weights), las conexiones que van desde la capa de entrada a la capa oculta [3 x 4] más las que van de esta hacia la capa de salida [4 x 2] y tiene 6 sesgos (biases) para total de 26 parámetros.

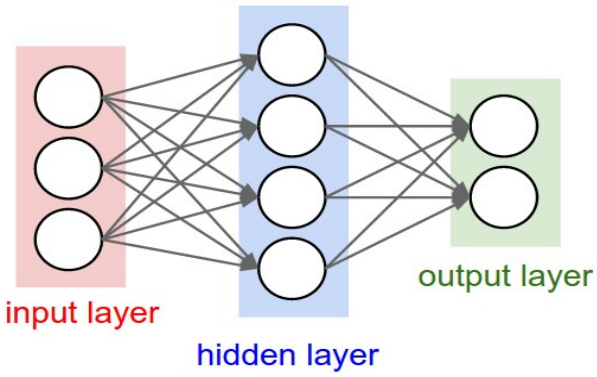


Figura 3.9: Red neuronal de 2 capas, 2-layer. Fuente: Johnson et al. [24]

3.5.2. Package "h2o", R Interface for H2O

El primer acercamiento en la búsqueda del mejor modelo se la realiza mediante el uso del paquete "h2o"[1], de R. Esta herramienta con intefáz en R es de código abierto y proporciona una plataforma de máquinas de aprendizaje rápida y escalable, compatible con varios modelos, entre ellos, "neural networks", (h2o.deeplearning). Se incia la herramienta con la instrucción "init", luego de hacer referencia a la librería h2o.

```
library(h2o)
h2o.init()
```

Este inicio estándar de la herramienta advierte que se limitará el rendimiento al uso de 2 cores de CPU por defecto.

```
##H2O is not running yet, starting it now...
##
##Note:  In case of errors look at the following log files:
##    C:\Users\oaruvi\AppData\Local\Temp\RtmpIZIMRE/h2o_oaruvi_...
##    C:\Users\oaruvi\AppData\Local\Temp\RtmpIZIMRE/h2o_oaruvi_...
##
##java version "1.7.0_79"
##Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
##Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
##
##Starting H2O JVM and connecting: . Connection successful!
##
##R is connected to the H2O cluster:
##    H2O cluster uptime:         2 seconds 323 milliseconds
##    H2O cluster version:        3.8.3.3
##    H2O cluster name:           H2O_started_from_R_oaruvi_dbm269
##    H2O cluster total nodes:    1
##    H2O cluster total memory:   2.65 GB
##    H2O cluster total cores:    4
##    H2O cluster allowed cores:  2
##    H2O cluster healthy:        TRUE
##    H2O Connection ip:          localhost
##    H2O Connection port:        54321
##    H2O Connection proxy:       NA
##    R Version:                  R version 3.3.1 (2016-06-21)
##
##Note: As started, H2O is limited to the CRAN default of 2 CPUs.
```

```
##      Shut down and restart H2O as shown below to use all your CPUs.
##         > h2o.shutdown()
##         > h2o.init(nthreads = -1)
```

En [28] se señalan ajustes para la obtención de un mejor rendimiento. El argumento "nthreads", que es opcional se relaciona con el número de CPUs utilizadas, (-2) para el valor por defecto de 2 CPUs y (-1) para utilizar todos las CPU disponibles en el host. El argumento "max_mem_size", que también es opcional especifíca con una cadena de caracteres el tamaño máximo de la memoria reservada para H2O, en bytes, megabytes o gigabytes.

```
h2o.init(ip="localhost", port=54321, max_mem_size="12g", nthreads=-1)
```

Este inicio optimizado permite el uso de 4 cores de CPU y un total de uso de memoria de 10,67 GB.

```
##H2O is not running yet, starting it now...
##
##Note:  In case of errors look at the following log files:
##    C:\Users\oaruvi\AppData\Local\Temp\Rtmp6zDvYb/h2o_oaruvi_...
##    C:\Users\oaruvi\AppData\Local\Temp\Rtmp6zDvYb/h2o_oaruvi_...
##
##java version "1.7.0_79"
##Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
##Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
##
##Starting H2O JVM and connecting: .. Connection successful!
##
##R is connected to the H2O cluster:
##    H2O cluster uptime:         4 seconds 478 milliseconds
##    H2O cluster version:        3.8.3.3
##    H2O cluster name:           H2O_started_from_R_oaruvi_blm823
```

```
##    H2O cluster total nodes:    1
##    H2O cluster total memory:   10.67 GB
##    H2O cluster total cores:    4
##    H2O cluster allowed cores:  4
##    H2O cluster healthy:        TRUE
##    H2O Connection ip:          localhost
##    H2O Connection port:        54321
##    H2O Connection proxy:       NA
##    R Version:                  R version 3.3.1 (2016-06-21)
```

La primera configuración de los parámetros de este modelo es de 5-layer, una capa de entrada de 784 neuronas (columnas del dataset), una primera capa oculta de 400 neuronas, una segunda de 200, una tercera de 2, una cuarta de 200 y una quinta de 400. Esta configuración aunque resulta no ser la más adecuada para la obtención de los resultados esperados, proporciona un ejemplo comprensible del modelo.

```
#Modelo Deep Learning h2o
modeloDeepLearning_h2o <- h2o.deeplearning(x = 2:785,
training_frame = train_h2o,
hidden = c(400, 200, 2, 200, 400 ),
epochs = 600,
activation = "Tanh",
autoencoder = TRUE)
```

La capa intermedia es una representación de 2 dimensiones de un dígito de 784 dimensiones. Se puede trazar una representación gráfica de esta bidimensionalidad en base a las características del modelo extraídas de la función "h2o.deepfeatures".

```
caracteristicas = h2o.deepfeatures(modeloDeepLearning, train_h2o,
layer=3)
dim(caracteristicas)
## [1] 42000   2
```

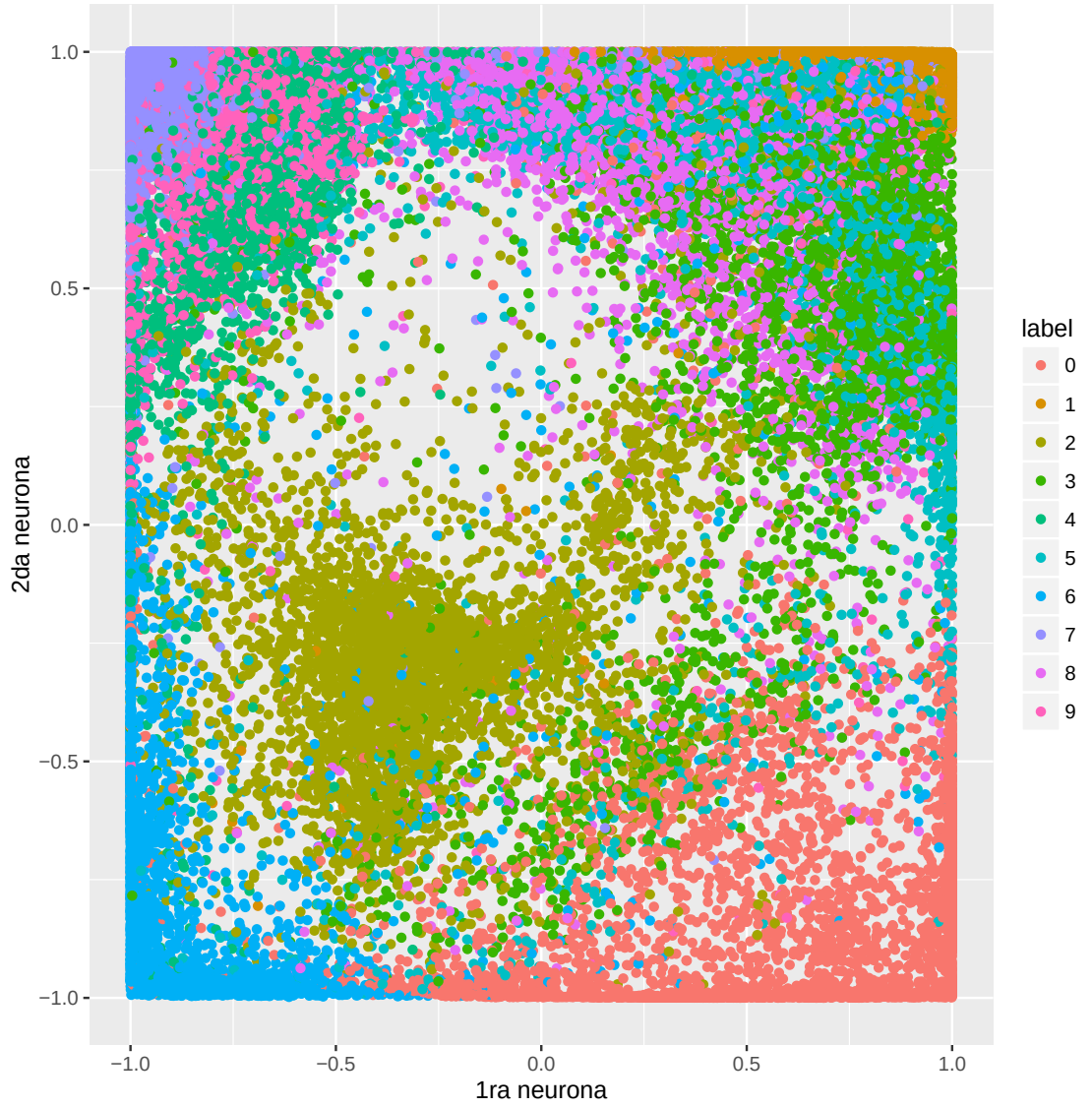


Figura 3.10: Red Neural de train_kaggle, neuronas: 400,200,2,200,400

La Figura 3.10 muestra la distribución de los dígitos del modelo Deep learning autoencoder, en donde la capa oculta central contiene una representación de 2 neuronas, que obliga al modelo a comprimir la información la cual se puede volver a descomprimir para recuperar la versión original a la salida. Para terminar el uso de la herramienta se utiliza la instrucción: "shutdown".

```
h2o.shutdown()

## Are you sure you want to shutdown the H2O instance running at
## http://localhost:54321/ (Y/N)?
## [1] TRUE
```

En [4] se describe a detalle la utilización de h2o y en [15] se expone la configuración con la que Arnol Candel obtiene el récord mundial con el conjunto de datos MNIST sin tratamiento previo. Esta configuración es la que se utiliza para realizar la experimentación con este modelo, de la siguiente manera:

```
#Modelo Deep Learning optimizado para la experimentación
modeloDeepLearning_h2o <- h2o.deeplearning(x = 2:dm,
y = 1,
training_frame = train_h2o,
activation = "RectifierWithDropout",
hidden = c(1024, 1024, 2048),
epochs = 8000,
l1 = 1e-5,
input_dropout_ratio = 0.2,
train_samples_per_iteration = -1,
classification_stop = -1)
```

Los parámetros del modelo son los siguiente:

- x: vector con las variables predictoras,
- y: variable a predecir,
- training_frame: dataset de tipo objeto H2OFrame,
- activation: cadena de caracteres que indica la función de activación,
- hidden: dimensión de las capas ocultas,
- epochs: número de veces de iteración del dataset,

- l1: regulariza y estabiliza el sistema,
- input_dropout_ratio: A fraction of the features for each training row to be omitted from training in order to improve generalization (dimension sampling).
- train_samples_per_iteration: número de muestras de training, (-1 disponible toda la data),
- classification_stop: criterio de parada en caso de error, (-1 desactivado).

3.5.3. Deep Learning con MXNet para R

MXNet es una librería de aprendizaje automático multilenguaje, que incluye a Python, R, y Julia, se ejecuta en la mayoría de los sistemas operativos y plataformas móviles, como Android. MXNet es ligero y portátil y ofrece eficiencia tanto en coste computacional como en el uso de memoria. El paquete MXNet de R permite construir y personalizar los modelos de aprendizaje adaptándolos para retos científicos de clasificación de imágenes y datos.

Una primera versión del modelo se compone de una capa de entrada de 784 neuronas (columnas del dataset), y las capas ocultas se componen de 128, 64 y 10 neuronas respectivamente la última corresponde a la salida necesaria de 10 dígitos. Se utiliza este modelo sólo como práctica para el entendimiento de la herramienta.

```
#Primera versión del modelo Deep Learning MNXnet
model <- mx.model.FeedForward.create(softmax, X=train.x, y=train.y,
ctx=devices, num.round=10, array.batch.size=100,
learning.rate=0.07, momentum=0.9,  eval.metric=mx.metric.accuracy,
initializer=mx.init.uniform(0.07),
epoch.end.callback=mx.callback.log.train.metric(100))
```

En LeCun et al. [17] se propone la utilización de redes neuronales convolucionales (LeNet) cuya estructura se asimila a las neuronas en la corteza visual primaria de un cerebro biológico. Este tipo de red son más efectivas para tareas de visión artificial, especialmente para el reconocimiento de dígitos escritos a mano y clasificación de imágenes en general. Esta red se construye de la siguiente manera:

Primera convolución:

```
conv1 <- mx.symbol.Convolution(data=data, kernel=c(2,2),
num_filter=20)
```

Segunda convolución:

```
conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(2,2),
```

Primera capa oculta con 500 neuronas:

```
fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
```

Segunda capa oculta con 10 neuronas (10 dígitos):

```
fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)
```

En "lenet", se obtiene la predicción probabilística:

```
lenet <- mx.symbol.SoftmaxOutput(data=fc2)
```

Parámetros de la configuración utilizada:

```
#Modelo Deep Learning MXNet optimizado para la experimentación
modeloDeepLearningMNXnet <- mx.model.FeedForward.create(lenet,
X=training_x_array, y=training_y,
ctx=device_cpu, num.round=60, array.batch.size=100,
learning.rate=0.1, momentum=0.5, wd=0.00001,
eval.metric=mx.metric.accuracy,
epoch.end.callback=mx.callback.log.train.metric(100))
```

3.5.4. Construcción de una red neuronal de 2 capas

Con la intención de profundizar en el estudio de la funcionalidad del deep learning se considera realizar la construcción de una red neural de 2 capas para realizar las experimentaciones, esta investigación se fundamenta en el curso de Redes Neuronales de Stanford [24].

Se presupone que con la normalización de los datos la mitad de los pesos será positiva y la otra mitad negativa, no conocemos el coste final de cada peso antes de ser entrenado, por lo que asignar un valor de cero a todos los pesos no es lo correcto eliminando la fuente de asimétria entre las neuronas. Se inicializan los pesos con valores aleatorios pequeños. Un problema con la sugerencia anterior es que la distribución de las salidas de una neurona inicializada al azar tiene una varianza que crece con el número de entradas.

```
W1 <- 0.01 * matrix(rnorm(dimension*clases), nrow = dimension)
b1 <- matrix(0, nrow = 1, ncol = clases)
W2 <- 0.01 * matrix(rnorm(clases*num_clases), nrow = clases)
b2 <- matrix(0, nrow = 1, ncol = num_clases)
```

Se obtiene la capa oculta de cada iteración con la función ReLU (Rectified Linear Unit) activada por defecto, esta función se expresada como: $f(x) = max(0, x)$.

```
capa_oculta <- pmax(0, training_x%*% W + matrix(rep(b,N), nrow = N,
byrow = T))
capa_oculta <- matrix(capa_oculta, nrow = N)
```

El cálculo interno de la red se obtiene multiplicando los valores de una neurona por los pesos de las conexiones salientes, y se suman los input que recibe de las conexiones entrantes.

```
puntaje <- capa_oculta%*%W2 + matrix(rep(b2,num_datos),
nrow = num_datos, byrow = T))
```

La función de pérdida de datos (loss), en aprendizaje supervisado mide la compatibilidad entre una predicción, es decir las puntuaciones de la clase en la clasificación

y la etiqueta verdadera. Expresado como: $L = \frac{1}{N}\sum_i L_i$, donde N es el número de datos de entrenamiento.

```
prob_log <- -log(probabilidad)
calc_loss <- sum(prob_log*Y)/num_datos
```

Para controlar la capacidad de las redes neuronales evitando el sobreajuste (overfitting) se utiliza "L2 regularization"que es quizás la forma más común de regularización. Se puede implementar penalizando la magnitud al cuadrado de todos los parámetros directamente en el objetivo. Por cada peso w en la red, se añade el término: $\frac{1}{2}\lambda w^2$, donde λ es la fuerza de regularización ingresada como parámetro en la función.

```
reg_loss <- 0.5*regularizar*sum(W1*W1) + 0.5*regularizar*sum(W2*W2)
```

Cálculo de loss:

```
loss <- calc_loss + reg_loss
```

En backpropagation se distinguen dos fases, en la primera se aplica un patrón de entrada, el cual se propaga por las capas hasta producir una salida. En la segunda fase el output se compara con la salida deseada y se cálcula el error cometido por cada neurona de salida. Estos errores se transmiten hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de las capas intermedias. Cada neurona recibe un error que es proporcional a su contribución sobre el error total de la red. Se ajustan los errores de los pesos. Para realizar estos cálculos en los parámetros de la red se obtienen previamente los gradientes correspondientes:

```
g_puntaje <- probabilidad-Y
g_puntaje <- g_puntaje/num_datos
```

Luego se calcula el gradiente del segundo peso $w2$ y sesgo:

```
g_W2 <- t(capa_oculta)%*%g_puntaje
g_b2 <- colSums(g_puntaje)
```

En la capa oculta se calcula el gradiente y la no linealidad ReLU:

```
g_capa_oculta <- g_puntaje%*%t(W2)
g_capa_oculta[capa_oculta <= 0] <- 0
```

Se calcula el gradiente del primer peso $w1$ y sesgo:

```
g_W1 <- t(training_x)%*%g_capa_oculta
g_b1 <- colSums(g_capa_oculta)
```

Se agrega la regularización a los pesos con gradiente:

```
g_W2 <- (g_W2) + regularizar * W2
g_W1 <- (g_W1) + regularizar * W1
```

En el entrenamiento de las redes neuronales profundas, un elemento a tener en cuenta es el crecimiento de la tasa de aprendizaje siendo alta el sistema se vuelve caótico y es incapaz de llegar a partes más profundas. Saber cuándo decaerá la tasa de aprendizaje puede ser complicado para lo cual se opta generalmente por realizar una reducción de la tasa de aprendizaje disminuída en una constante (por ejemplo 0,5). Sobre la base de esta aclaración finalmente se actualizan los parámetros de salida de la función, esto se consigue restando de los pesos y sesgos el producto de la tasa de aprendizaje, valor ingresado como parámetro de la función, por los gradientes obtenidos.

```
W1 <- (W1) - tasa_aprend * g_W1
b1 <- (b1) - tasa_aprend * g_b1
W2 <- (W2) - tasa_aprend * g_W2
b2 <- (b2) - tasa_aprend * g_b2
```

El Algoritmo 4 describe los pasos realizados en la construcción de una red neuronal de 2 capas, recibe 6 parámetros de entrada descritos a continuación:

- training: dataset train,
- testing: una matriz formada por el número de filas de training y número de clases a predecir,
- tasa_aprend: valor constante de tasa de aprendizaje,
- regularizar: valor constante de factor de regularización,
- clases: número de clases a predecir,
- n_veces: número de veces que se realiza el cálculo.

Algoritmo 4 Construcción de Red Neuronal de 2 capas

Entrada: Seleccionar training, testing, tasa_aprend, regularizar, clases, n_veces

Salida: Lista de parámetros: W1, b1, W2, b2

1: Obtener las dimensiones de los datos de entrada: (num_datos, num_clases, dimensión)
2: Inicializar los parámetros de la red al azar: (W1, b1, W2, b2)
3: **para** i = 1 hasta n_veces **hacer**
4: Calcular la capa oculta con la función ReLU
5: Calcular la puntuación de la clase
6: Calcular las probabilidades de clase y normalizarlas
7: Calcular la función loss y aplicar la regularización
8: Calcular el gradiente de puntaje
9: Calcular backpropagation y gradientes de los parámetros
10: Calcular de regularización en los pesos
11: Actualizar los parámetros de salida con la tasa de aprendizaje
12: **fin para**
13: **devolver** Lista(W1, b1, W2, b2)

Capítulo 4

Resultados

El objetivo de este capítulo es presentar los resultados de la experimentción realizada en el transcurso del presente trabajo. Sobre la base del conocimiento de los dataset tanto de MNIST como de la competencia Kaggle, se realizó la investigación teórica que sustenta la definición de herramientas y técnicas adecuadas para lograr la consecución de los objetivos planteados.

Se presentan primero los resultados del análisis predictivo de dígitos manuscritos, los preprocesos utilizados, la identificación y configuración de los modelos de aprendizaje automático y los resultados de la experimnetación realizada. Se detalla además los resultados de la participación propia y de otros participantes en la competencia de Kaggle: Digit Recognizer. En el Cuadro 4.1 se describe el computador utilizado en la experimentación.

Características de computador	
Procesador	Intel® Core™ i5-5200U
Frecuencia básica del procesador	2.2 GHz
Cantidad de núcleos	2
Cantidad de subprocesos	4
Memoria RAM	12,0 GB
Tipo de sistema	64-bits
Sistema Operativo	Windows 10 Home

Cuadro 4.1: Computador usado en los cálculos

4.1. Experimentos con los modelos predictivos

Se procura que la experimentación con los modelos predictivos sea en las mismas condiciones, para esto se fija la misma semilla[1] en todas las pruebas y se realiza el mismo procedimiento. Teniendo en cuenta que el dataset test proporcionado por Kaggle no contiene la columna de etiqueta se hace difícil la comprobación de la predicción realizada en la experimentación. Se crea por lo tanto una muestra a partir del dataset train para obtener nuevos conjuntos de training y testing que contengan ambos la etiqueta del dígito a predecir. Para estas pruebas en particular se obtienen un subconjunto de 10.000 dígitos para testing y otro de 32.000 dígitos para training. El Cuadro 4.2 muestra la matriz de confusión obtenida en el modelo MXNet con el dataset **train_kaggle_bo** mejor valoración de toda la experimentación. El accuracy se cálcula con la sumatoria de la diagonal principal de la tabla y dividiendo para el total de valores predichos. Un total de 9.882 dígitos bien clasificados en una muestra de 10.000 dígitos, siendo la exactitud de predicción de este modelo de **98,82 %**.

		Predicción										
		0	1	2	3	4	5	6	7	8	9	Error
Etiqueta	0	981	0	1	0	0	0	7	0	0	2	0,01009
	1	0	1092	0	1	0	0	0	1	1	0	0,00274
	2	1	3	1033	2	2	0	0	4	0	0	0,01148
	3	1	0	4	1001	0	1	0	0	1	1	0,00793
	4	0	1	0	0	952	0	2	0	0	12	0,01551
	5	0	0	0	7	0	895	3	0	1	0	0,01214
	6	2	0	0	0	1	2	996	0	2	0	0,00698
	7	0	5	9	1	0	0	0	1016	1	7	0,02214
	8	2	0	4	0	2	2	0	1	936	3	0,01474
	9	3	0	0	1	4	2	0	4	1	980	0,01508
	Tot	990	1101	1051	1013	961	902	1008	1026	943	1005	0,01180

Cuadro 4.2: Matriz de confusión

Las filas indican la etiqueta real del dígito mientras que las columnas los dígitos

[1] En R se utiliza la función set.seed( ) para obtener números aleatorios reproducibles, el argumento es un número entero cualquiera.

predichos, así el dígito 0 tiene 981 aciertos, se clasifica erróneamente una vez en confusión por el dígito 2, siete veces por el dígito 6, y dos veces por el dígito 9, la columna error resulta de la razón entre el total de veces que el dígito estuvo mal predicho (10 para el dígito 0, primera fila), y el total de ocurrencias de la etiqueta real del dígito (991 para el dígito 0, primera fila). El error de prueba de este modelo es de **1,18 %**.

En el capítulo anterior se obtuvieron algunos dataset con diferente tipo de tratamiento previo, el Cuadro 4.3 muestra el resumen del preprocesamiento realizado con el dataset train oiginal de Kaggle. Se puede identificar el tratamiento realizado, el nombre del dataset resultante con la descripción de sus dimensiones y el tiempo empleado en el proceso expresado en segundos.

Preprocesamiento del dataset train de Kaggle			
Tratamiento	Dataset Resultante	Dimensión	Tiempo (seg.)
ninguno	train_kaggle	42000 x 785	0,00
int	train_kaggle_int	42000 x 786	4,57
red	train_kaggle_r	42000 x 197	110,17
bin	train_kaggle_bo	42000 x 785	288,19
red + bin	train_kaggle_br	42000 x 197	162,91
bin + int	train_kaggle_boi	42000 x 786	292,76
red + bin + int	train_kaggle_bri	42000 x 198	167,48
bin + pca	train_kaggle_pca_bo	42000 x 65	315,68
red + bin + pca	train_kaggle_pca_br	42000 x 65	164,96

bin = binarización (umbralización por método de Otsu)
red = reducción (transformada Wavelet de Haar)
int = intensidad (aumento de nueva característica)
pca = análisis de componentes principales

Cuadro 4.3: Resultados del preprocesamiento

Tres son los modelos predictivos utilizados para la experimentación, el primero h2o con su implementación de Deep Learning ofrece escalabilidad y rapidez con configuraciones que permiten el multiprocesamiento, es decir el uso de dos o más procesadores o CPUs, y entre otras, la posibilidad de especificar el tamaño de me-

moria, aspectos importantes que hacen de esta herramienta una de las más utilizadas en problemas de clasificación y con suficiente documentación disponible [9].

La versión y configuración utilizada para estas pruebas permiten el uso de máximo 4 unidades de procesamiento con un tamaño de memoria de 12 GB, pero las restricciones de hardware de la máquina utilizada permite un máximo de 2 CPUs y un máximo de 10,67 GB, lo que hace que los tiempos para este modelo no sean de los mejores presentados.

El Cuadro 4.4 muestra el resumen de la experimentación en donde se identifica el modelo predictivo, el dataset insumo para los cálculos, la exactitud de predicción, el tiempo empleado y el tiempo total que consolida los tiempos globales tomando en cuenta el tiempo de preproceso.

Experimentación del modelo predictivo con los dataset obtenidos				
Modelo	Dataset	Accuracy (%)	Tiempo (seg.)	Tiempo Total (seg.)
h2o	train_kaggle	98,41	22371,26	22371,26
	train_kaggle_bo	98,22	18331,55	18619,74
	train_kaggle_br	97,59	26224,96	26387,87
	train_kaggle_boi	98,09	15515,57	15808,33
	train_kaggle_bri	97,53	27115,42	27282,90
	train_kaggle_pca_bo	98,49	15027,91	15343,59
	train_kaggle_pca_br	97,68	18342,16	18507,12
MXNet	train_kaggle	98,04	4291,57	4291,57
	train_kaggle_bo	98,82	4875,65	5163,84
	train_kaggle_br	97,15	879,83	1042,74
	train_kaggle_pca_bo	81,89	416,20	731,88
	train_kaggle_pca_br	85,60	413,43	578,39
RN_2l	train_kaggle	93,55	806,36	806,36
	train_kaggle_bo	94,33	1025,34	1313,53
	train_kaggle_br	91,12	369,55	532,46

Cuadro 4.4: Resultados de la experimentación del análisis predictivo

El segundo modelo MXNet dispone de una flexible y eficiente librería para Deep Learning [19], se ejecuta en las CPU o GPU[2], en los clústeres, servidores, ordenadores de sobremesa, o teléfonos móviles. Este modelo evidencia los mejores resultados en tiempo de ejecución con el mismo hardware. Y por último el tercer modelo es una implementación sencilla codificada en R de una red neuronal 2-layer en la que se pretende realizar un ensayo didáctico enfocado en el conocimiento más que en obtener los mejores resultados.

4.2. La competencia de Kaggle: Digit Recognizer

El archivo de resultados para la presentación en la competencia de Kaggle se obtiene utilizando el modelo y preproceso con el mejor resultado del Cuadro 4.4, esto es el modelo MXNet con el dataset binarizado que obtuvo el registro de 98,82 % con un tiempo de 5163,84 segundos. El archivo para la competencia tiene el siguiente formato:

```
dim(resultados)

## [1] 28000   2

head(resultados)

##   ImageId Label
## 1       1     2
## 2       2     0
## 3       3     9
## 4       4     9
## 5       5     3
## 6       6     7
```

El Cuadro 4.5 muestra el resultado de la participación en la competencia, alcanzando un **99,129 %** de efectividad en la predicción de dígitos manuscritos para el dataset test proporcionado por Kaggle, el tiempo total para el proceso de este

[2]Coprocesador dedicado al procesamiento de gráficos que aligera la carga de trabajo del procesador central

resultado es de 6940,21 segundos, tiempo total del preproceso y la predicción. Con este porcentaje se alcanza la posición **128** de **1044** participantes en la fecha **30 de agosto de 2016**, top **12 %** en la competencia, Figura 4.1.

Resultados de la competencia de Kaggle					
Dim. test	Preproceso	Tiempo (seg.)	Modelo	Accuracy (%)	Tiempo (seg.)
28000 x 784	binarización	288,19	MXNet	99,129	6940,212

Cuadro 4.5: Resultados de la experimentación del análisis predictivo

La tabla de posiciones de la competencia de Kaggle: Digit Recognizer se puede observar en: https://www.kaggle.com/c/digit-recognizer/leaderboard. Esta tabla de clasificación se calcula con aproximadamente el 25 % de los datos de prueba. Los resultados finales se basarán en el 75 % restante, por lo que la clasificación final puede ser diferente, según se explica en la página de la competencia.

Figura 4.1: Puesto obtenido en la competencia de Kaggle: Digit Recognizer

El top ten de los mejores resultados se muestran en la Figura 4.2, existen tres participantes que han obtenido la máxima puntuación, el resto separado por apenas décimas que evidencian la competitividad de las entradas, en muchos de los casos se desconoce los preproceso y modelos utilizados por los participantes. La Figura 4.3 representa los resultados de la predicción de los 100 primeros dígitos del dataset test de Kaggle.

#	Δ1w	Team Name	Score	Entries	Last Submission UTC (Best – Last Submission)
1	–	Riken Mehta	1.00000	3	Sat, 23 Jul 2016 14:24:41
2	–	AlexKH	1.00000	7	Mon, 15 Aug 2016 00:00:40
3	–	Gang of Three	1.00000	29	Sat, 03 Sep 2016 12:31:52 (-18.8d)
4	–	vyujing	0.99971	3	Thu, 04 Aug 2016 14:39:34
5	–	JeffreyXiao	0.99957	5	Sun, 07 Aug 2016 14:07:41
6	–	Andre lopes	0.99957	1	Sun, 07 Aug 2016 19:31:42
7	–	zhxfl	0.99957	1	Mon, 15 Aug 2016 02:50:48
8	–	li bin	0.99943	15	Sat, 06 Aug 2016 05:44:03 (-20.1d)
9	–	XuleiYang	0.99943	1	Thu, 28 Jul 2016 01:10:14
10	↑390	meurich	0.99929	4	Mon, 05 Sep 2016 18:59:12

Figura 4.2: Top Ten competencia de Kaggle: Digit Recognizer

Considerando el resultado existen un error de predicción de 0.871 %, en la Figura 4.3 (pos: 1x4) la predicción muestra el dígito nueve cuando en realidad es cero.

Figura 4.3: Predicción de los primeros 100 dígitos del dataset test

Capítulo 5

Conclusiones y trabajo futuro

Luego de realizar la experimentación con los diferentes dataset obtenidos en la fase de análisis y preproceso, de utilizar algunos modelos predictivos de comprobada efectividad y de participar en la competencia de Kaggle, se llega a las siguientes conclusiones.

- El análisis previo que se realizó al dataset train de Kaggle hace referencia a la intensidad de escritura de cada dígito, el dígito cero es aquel con más intensidad de trazo visible, y el dígito uno el de menos intensidad, esta carácterística se adicionó como nuevo elemento en dos dataset. Para el dataset train_kaggle_boi se obtuvo como mejor resultado el 98,09 % y para el dataset train_kaggle_bri el 97,53 % de efectividad predictiva en el modelo h2o. Con relación a los resultados de los dataset de las mismas características no mejora la efectividad predictiva en ninguno de los dos casos.

- El preproceso de reducción se lo combina con la binarización para obtener dos dataset. Para el dataset train_kaggle_bo se obtuvo como mejor resultado el 98,82 % en el modelo MXNet y para el dataset train_kaggle_br el 97,59 % en el modelo h2o. El dataset original binarizado obtiene el mejor resultado de toda la experimentación y es utilizado para la participación en la competencia de Kaggle. Binarizar la imagen sobre la base de la umbralización óptima constituye el mejor preproceso de la presente investigación y se evidencia en las imágenes producidas, siendo estas más nítidas que las originales. La reducción no mejora

los resultados obtenidos con el dataset original esto puede ser comprensible desde el hecho que visualmente se nota la diferencia en las imágenes producidas por cada dataset.

- La selección de variables basada en PCA obtiene dos dataset, con el modelo h2o se alcanza 98,49 % para train_kaggle_pca_bo y 97,68 % para train_kaggle_pca_br de efectividad, para el modelo MXNet los valores de 81,89 % y 85,60 % respectivamente, siendo estos últimos los más bajos valores alcanzados en toda la experimentación debido a que se utilizó una configuración estándar del modelo que no beneficia particularmente a las características de estos dataset.

- Se realiza la construcción de un modelo de red neuronal de 2 capas, si bien es cierto en la experimnetación los valores obtenidos por este modelo no superan a los frameworks de modelos predictivos, el objetivo de esta actividad es obtener un mejor entendimiento del funcionamiento de las redes neuronales. Se alcanzan valores de efectividad predictiva adecuados en tiempos bajos considerando las restricciones de funcionalidad del modelo.

- Se realizó el análisis de dos modelos predictivos provistos por librerías de código abierto para la implementación de deep learning. El modelo h2o se utilizó en la presente investigación de manera didáctica, obteniendo la suficiente experimentación entre prueba y error para hacer funcionar esta librería y obtener cada vez los mejores resultados, este modelo en comparación tiene su debilidad en el coste computacional elevado provocado en buena parte por la limitante del hardware empleado en la experimentación. El modelo MXNet forma parte del top 10 de proyectos punteros de deep learning [14], luego de una correcta configuración alcanza similares resultados que h2o pero a un coste computacional sin comparación, constituyéndose el modelo con los mejores resultados de la presente investigación.

- El porcentaje de efectividad predictiva (accuracy) es alto para toda la experimentación, encontrando paridad en los modelos seleccionados. El mejor resultado global de las pruebas es de 98,82 %, y la calificación obtenida en la participación de la competencia de Kaggle: Digit Recognizer es de 99,129 % de

efectividad de predicción para datos reales. Esta calificación da como resultado un margen de error del 0,871 %.

- Se cumplen los objetivos de la presente investigación, se realiza la identificación efectiva del dígito de una imagen que contiene un sólo número escrito a mano en un porcentaje de efectividad del 99,129 %, se utiliza la técnica de umbralización óptima de imágenes por el método de Otsu para lograr obtener el mejor porcentaje de efectividad predictiva en un tiempo de 5183,64 segundos con el modelo MXNet. Se cumple la hipótesis propuesta de alcanzar scores similares a los publicados en la competencia de Kaggle, la cual es de altísimo nivel produciendo resultados que se superan constantemente en el transcurso del tiempo de competencia.

Para la continuación de trabajos futuros sobre la base de la presente investigación, se recomienda concentrarse sobre dos aspectos importantes del análisis predictivo. El primero es acerca del tratamiento previo que se le aplique al dataset, al profundizar sobre este particular, se considera que los resultados de la clasificación de dígitos pueden mejorar, resulta evidente que mejorando la apariencia de algunas de las imágenes que aún para el ojo humano resultan díficiles de interpretar se pueda lograr una mejora en el aprendizaje del modelo y la predicción resultante. Una técnica que resulta interesante de aplicar sería la mencionada en [23] y [7] sobre la rotación de los dígitos, de tal manera que se logre ubicar su referencia horizontal perpendicular a su referencia vertical con su centro de gravedad centrado.

El segundo aspecto es sobre profundizar en técnicas para el mejoramiento del tiempo de proceso y configuración de parámetros para los modelos predictivos estudiados, en [28] se detalla como mejorar el rendimiento de h2o y en [9] y [19] los APIs de referencia de h2o y MXNet respectivamente, pero sin duda exiten muchos más sitios especializados en lograr estos objetivos. Por último resaltar que se puede mejorar el rendimiento computacional de los modelos con la utilización de hardware enfocado al procesamiento paralelo y la incorporación de las bondades que presenta el uso de la GPU para cálculos de coma flotante [1] en el procesamiento de imágenes.

[1] Se utiliza para representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta

Bibliografía

[1] Aiello S., Kraljevic T. y Maj P. (2016, julio 13) Package "h2o". *https://cran.r-project.org/web/packages/h2o/h2o.pdf*, 2016.

[2] Borbón A. y Mora W. Edición de Textos Científicos Latex 2014. *Revista digital Matemática, Educación e Internet. (http://tecdigital.tec.ac.cr/revistamatematica/)*, 2014.

[3] Bottou L. y LeCun Y. Graph transformer networks for image recognition. *Proceedings of ISI.*, 2005.

[4] Candel Arno. (2014, julio 16). Deep Learning with H2O [Mensaje en Blog]. *Recuperado de: http://www.slideshare.net/0xdata/h2o-distributed-deep-learning-by-arno-candel-071614*.

[5] Candel Arno. (2014, noviembre 21). Classification and Regression with H2O Deep Learning [Mensaje en Blog]. *Recuperado de: https://github.com/h2oai/h2o-training-book/blob/master/hands-on.training/deep.learning.md*, 2014.

[6] Carmona Francesc. (2014, enero 13)Francesc Carmona. Un ejemplo de ACPacp paso a paso. *Recuperado de: http://www.ub.edu/stat/docencia/Mates/ejemploACP.PDF*, 2014.

[7] Cireşan D., Meier U., Gambardella L. M., Schmidhuber J. Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition. *IEEE*, 2012.

[8] Google Brain Team. TensorFlow is an Open Source Software Library for Machine Intelligence [Mensaje en Blog]. *Recuperado de: https://www.tensorflow.org/*.

[9] H2O Community. H2O and Sparkling Water Documentation [Mensaje en Blog]. *Recuperado de: http://docs.h2o.ai/h2o/latest-stable/index.html*.

[10] Kaggle. Digit Recognizer. *Recuperado de: https://www.kaggle.com/c/digit-recognizer*.

[11] Khitalishvili Koba. (2015, noviembre 18). Digit recognizer in Python using CNN [Mensaje en Foro]. *Recuperado de: https://www.kaggle.com/kobakhit/digit-recognizer/digit-recognizer-in-python-using-cnn*.

[12] Khitalishvili Koba. (2015, noviembre 19). Digit Recognizer in R [Mensaje en Foro]. *Recuperado de: https://https://www.kaggle.com/kobakhit/digit-recognizer/digital-recognizer-in-r/versions*.

[13] Kirill Kliavin. (2016, febrero 22). TensorFlow deep NN [Mensaje en Foro]. *Recuperado de: https://www.kaggle.com/kakauandme/digit-recognizer/tensorflow-deep-nn*.

[14] KDnuggets. (2016, enero 20). Top 10 Deep Learning Projects on Github [Mensaje en Blog]. *Recuperado de: http://www.kdnuggets.com/2016/01/top-10-deep-learning-github.html*.

[15] KDnuggets. (2015, enero 28). Interview: Arno Candel, H20.ai on How to Quick Start Deep Learning with H2O [Mensaje en Blog]. *Recuperado de: http://www.kdnuggets.com/2015/01/interview-arno-candel-h20-deep-learning.html*.

[16] LeCun Y., Cortes C., y Burges C. The mnist database of handwritten digitsu. *Recuperado de: http://yann.lecun.com/exdb/mnist/www.iaci.unq.edu.ar*, 201605.

[17] LeCun Y., Bottou L., Bengio Y. yYann LeCun, Léon Bottou, Yoshua Bengio, y Patrick Haffner P. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[18] López Rubén. (2014, mayo 7). ¿Qué es y cómo funciona "Deep Learning"? [Mensaje en Blog]. *Recuperado de: https://rubenlopezg.wordpress.com/*, 2014.

[19] MXNet Community. Flexible and Efficient Library for Deep Learning [Mensaje en Blog]. *Recuperado de: https://mxnet.readthedocs.io/en/latest/.*

[20] Nunes Matt. (2014, abril 24) Package "binhf". *https://cran.r-project.org/web/packages/binhf/binhf.pdf*, 2014.

[21] O'Connor Brendan. Load the mnist data set in r. *Recuperado de: https://gist.github.com/brendano/39760*, 2008.

[22] Rodríguez Jorge. Método otsu. *Recuperado de: https://es.scribd.com/doc/45875911/Metodo.Otsu/*, 2010.

[23] Simard, P. Y., Steinkraus, D. y Platt, J. C. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Document Analysis and Recognition. *IEEE*, 2003.

[24] Johnson Justin y Karpathy Andrej . Cs231n: Convolutional neural networks for visual recognition. *Recuperado de: http://cs231n.github.io/*, 2016.

[25] Universidad Nacional de Quilmes. Segmentación por umbralización - método de otsu. *Recuperado de: www.iaci.unq.edu.ar*, 2005.

[26] Wikipedia. (2016, junio 3). Wavelet de haar [Mensaje en Blog]. *Recuperado de: https://es.wikipedia.org/wiki/Wavelet.de.Haar/*, 2015.

[27] Wikipedia. (2016, agosto 9). Análisis de componentes principales [Mensaje en Blog]. *Recuperado de: https://es.wikipedia.org/wiki/Análisis.de.componentes.principales/*, 2016.

[28] WPengine. (2015, febrero 27). The Definitive Performance Tuning Guide for H2O Deep Learning. *Recuperado de: http://blog.h2o.ai/2015/02/deep-learning-performance/.*