Efectos de la reducción de la dimensionalidad en la detección de dígitos manuscritos

Valcarce Ríos, Diego d.valcarce@udc.es
Teodoro Rodrigues, Joao Víctor victor.teodoro@udc.es

Abstract: Cuando se habla de clasificación muticlase, uno de los primeros problemas que suele venir a la mente es el reconocimiento de caracteres escritos, una cuestión ubicua en el mundo de la inteligencia artificial. En este paper atacaremos el problema haciendo uso de redes neuronales, con distintas topogías, aplicaremos reducción de la dimensionalidad empleando PCA, y respaldaremos los resultados por validación cruzada.

Palabras clave: Reconocimiento de dígitos, Redes Neuronales, Análisis de Componentes Principales, K-Fold Crossvalidation.

1. Introducción

El problema presentado consiste en la clasificación de dígitos numéricos manuscritos; el reto será distinguir los dígitos aún cuando existen infinitas posibles caligrafías; por lo que se hará uso del machine learning.

Obtener una solución a este problema puede ser clave para poder implementar sistemas inteligentes con el fin de automatizar tareas, como bien pudiera ser la digitalización de documentos o incluso la detección de caracteres en otro tipo de imágenes como sistemas para lectura de matrículas de vehículos. También podría ser relevante en el campo de investigación de la detección de patrones en imágenes.

2. Descripción del problema

Para aplicar las técnicas que serán descritas más adelante, es necesario conocer *a priori* qué requisitos se han utilizado a la hora de afrontar este problema, pues dependiendo de los mismos se obtendrán resultados diferentes.

En este caso concreto, se han utilizado imágenes manuscritas de un tamaño de 28x28 píxeles, en trazo negro sobre fondo blanco, como se muestra en la figura 1.

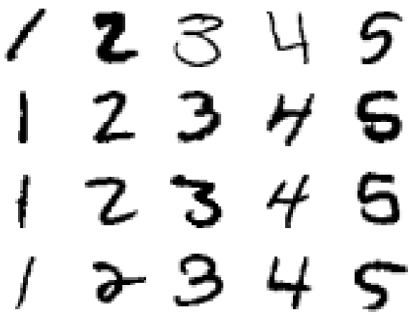

Figura 1: Imágenes de los dígitos

2.1. Descripción de la base de datos

La base de datos [1] cuenta con 42.000 observaciones de imágenes de dígitos escritos a mano por miembros de la Oficina del Censo de los Estados Unidos. En ella encontramos 785 atributos, siendo el primero de ellos el dígito al que pertenece cada patrón (entre 0 y 9) y los 784 restantes los píxeles de la imagen 28×28 que conforman la imagen original recolocados por fila y por columna.

Los píxeles toman valores comprendidos entre 0 y 255 en una escala de grises, donde 0 representa el blanco y 255 el negro.

3. Análisis Exploratorio de Datos

Empezaremos viendo la distribución de los dígitos que tenemos en el dataset. Esto nos

[1] Este conjunto de datos ha sido extraído de la página: http://www.kaggle.com/c/digit-recognizer.

permitirá ver si hay alguna clase que esté infrarrepresentada respecto a las demás.

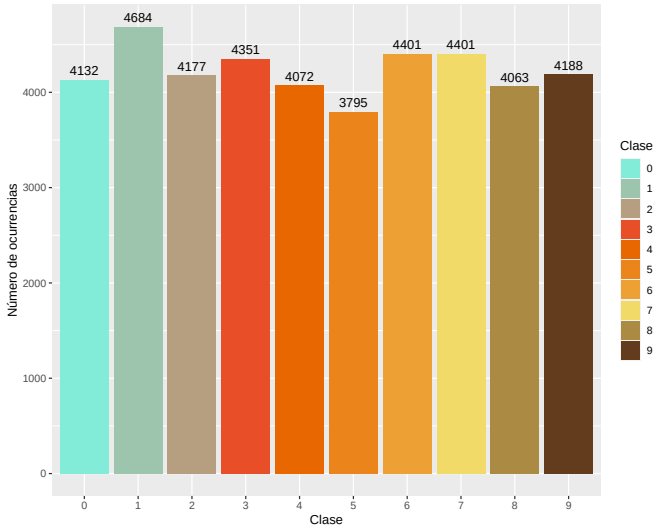

Figura 2: Frecuencias de los dígitos en la base de datos

Viendo los resultados de la gráfica, podríamos concluir que parece ser que no hay grandes diferencias entre el número de observaciones de cada dígito, y que estarían uniformemente distribuidos en nuestro conjunto de datos.

Para análisis posteriores es importante analizar dónde se encuentra la información en la imagen, es decir, dónde los píxeles están "activados" y dónde lo hacen con mayor intensidad. Para ello realizamos un mapa de calor de la información, como se observa en la figura 3. Podemos apreciar que la totalidad de los números se concentran en el centro de la imagen y no hay información (trazado o píxeles activos) en los bordes.

Esto es de tal medida que en el gráfico de sectores 4 observamos un desbalanceo total entre las clases píxel encendido o apagado (valores distintos de cero o cero, trazado o no trazado). Esta información será muy relevante a la hora de realizar el preprocesamiento de los datos (sección 5) aplicando técnicas de reducción de la dimensionalidad.

3.1. t-SNE

Una técnica interesante a la hora de visualizar datos de alta dimensionalidad puede ser el "**t-SNE**" (*t-Stochastic Neighbor Embedding*), introducido por Van der Maaten and Hinton,

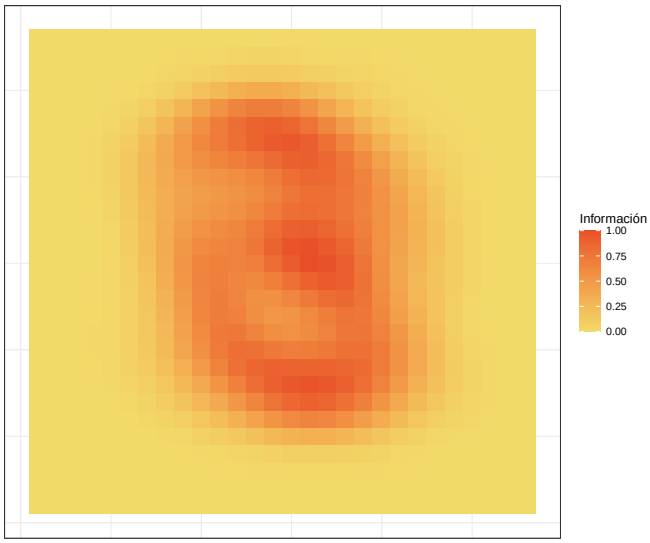

Figura 3: Mapa de Calor de la información

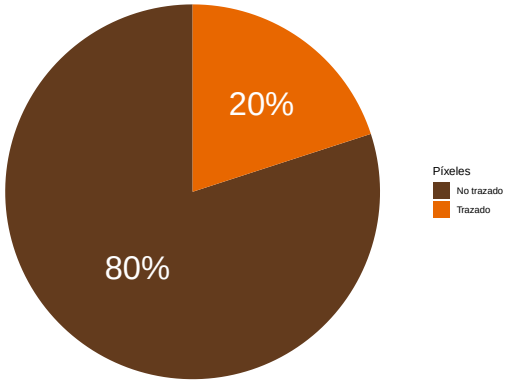

Figura 4: Presencia-ausencia de trazado

2008, como una mejora al ya existente modelo "SNE" (*Stochastic Neighbor Embedding*) de Hinton and Roweis, 2002, utilizando una distribución t de Student en vez de la guassiana para modelizar los datos.

Este método trata de conseguir proyecciones de los datos originales de alta dimensión en un plano 2D o espacio 3D, facilitando así la representación de los mismos; lo interesante reside en que es capaz de mantener la estructura local y además la estructura global, como la presencia de clústers. Es decir, mantiene las similitudes entre los miembros de un mismo grupo y las diferencias que existen entre las agrupaciones.

Como se observa en la figura 5, existe la

separación espacial entre los distintos dígitos en lo que a la configuración de píxeles se refiere, por lo que parece que *a priori* podemos esperar obtener resultados buenos.

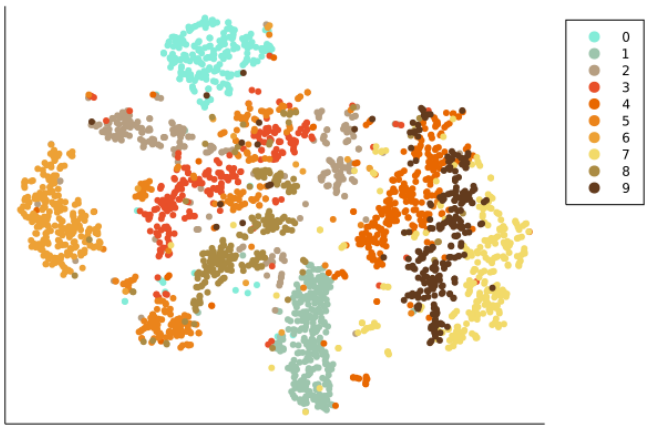

Figura 5: Representación de los datos usando la técnica t-SNE

De la misma forma, podemos realizar una estimación kernel de la densidad bidimensional (figura 6) de los datos obtenidos con t-SNE para ver dónde se concentran las modas de los valores que toma cada dígito, y observar que estas están bien separadas entre sí, exceptuando quizás el caso del 4 y el 9.

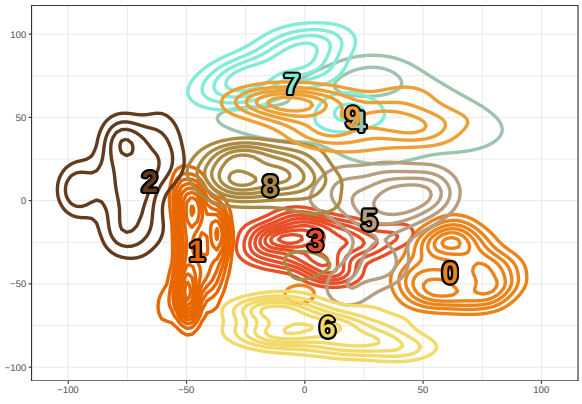

Figura 6: Representación de la densidad de los datos con t-SNE

4. Análisis bibliográfico

La primera aproximación más lógica podría ser la utilización de redes convolucionales (CNN, *Convolutional Neural Networks*), como en los trabajos realizados por Ciresan et al., 2011, con la técnica de *committee*, la cual consiste en entrenar distintas redes neuronales. En esta investigación, se utilizó la base de datos del MNIST completa, que introduce también caracteres, obteniendo así un total de más de 800000 observaciones, para las cuales se consiguió una ratio de error del 0.81% sobre los dígitos y un 11.88% al clasificar todos los caracteres alfanuméricos.

El trabajo de Babu et al., 2014, afronta el problema utilizando la técnica $k-$NN (*k-nearest neighbours*), concluyendo un *accuracy* del 96%. Esta misma precisión la logra LeCun et al., 1990, pero enfocando el trabajo de otra forma; en este estudio tratan de hacer más hincapié en realizar redes neuronales que se fijen en zonas locales en vez de en el patrón completo del dígito. Para llevarlo acabo, una de las soluciones propuestas es aplicar primeramente una red convolucional que tenga un tamaño de kernel reducido.

No obstante, en nuestro trabajo trataremos de no enfocarnos en obtener valores de métricas óptimas, sino que intentaremos balancear entre éstas y la complejidad del problema, es decir, trataremos de simplificar al máximo nuestro problema, pero sin llegar a comprometer los valores mencionados.

5. Preprocesado de los datos

Para facilitar el aprendizaje de nuestra red neuronal, los valores *input* se hallan normalizados y los valores *target*, codificados en formato *one-hot-encoding*. En la figura 7, se observa qué implicaciones tiene, a nivel visual, el normalizado de los píxeles de las imágenes (a la izquierda la imagen original, y a la derecha la normalizada).

Ante la gran dimensionalidad de nuestra base de datos, trataremos de aplicar técnicas de reducción de la misma, como PCA, en los siguientes apartados.

5.1. Motivación de la reducción de la dimensionalidad

En este subconjunto de datos que hemos seleccionado, contamos, como se ha mencionado en la sección 2.1, con 42000 observaciones y 784 variables, lo que implica que tenemos un total de 32928000 valores, por lo que

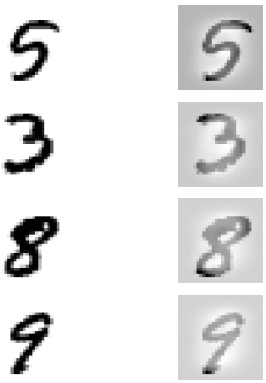

Figura 7: Resultado de normalizar los dígitos

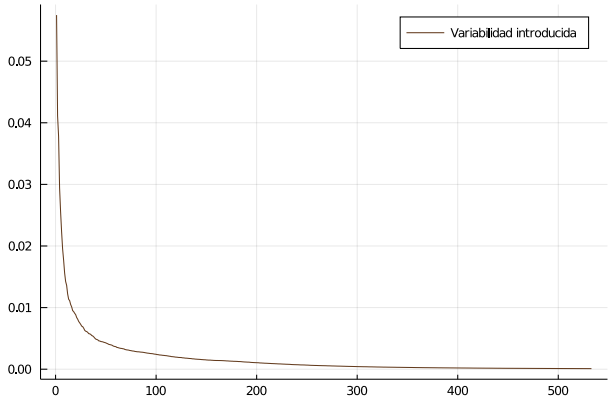

Figura 8: Gráfica del codo

como cabe suponer, ralentiza considerablemente la tarea de entrenamiento y de evaluación de nuevas observaciones. Como objetivo en esta sección, trataremos de afrontar este problema mediante técnicas de reducción de la dimensionalidad, concretamente PCA, y mencionaremos las ventajas y compromisos que esta técnica generará en la siguiente subsección.

5.2. PCA

Técnica introducida por Pearson, 1901, busca crear proyecciones ortogonales con la intención de explicar la mayor variabilidad posible con el menor número de lo que a partir de ahora llamaremos componentes (los vectores de proyección). Trataremos de encontrar un subconjunto de componentes tal que expliquen de forma prácticamente idéntica toda la información de la que dispondríamos si trabajásemos con la base de datos con todos los atributos.

Una forma de seleccionar el número de componentes principales reside en utilizar el gráfico del codo (figura 8), donde cada punto explica la variabilidad introducida en la proyección, al ser incluida la $i-$ésima componente principal. Las conclusiones se deben obtener respondiendo a: ¿a partir de qué momento se deja de incrementar la variabilidad explicada de forma suficientemente notoria? I.e., ¿en qué punto se "dobla el codo"?

En el caso de nuestra base de datos, esto se da en algún lugar de entre las primeras 100 componentes principales, conclusión que además podremos esclarecer con la figura 9, que muestra los resultados de entrenar una red neuronal con una única capa oculta, con el 90% de las neuronas de entrada (que viene dado por el número de componentes principales escogidas en cada uno de los entrenamientos). Las redes se han entrenado en uno contra todos.

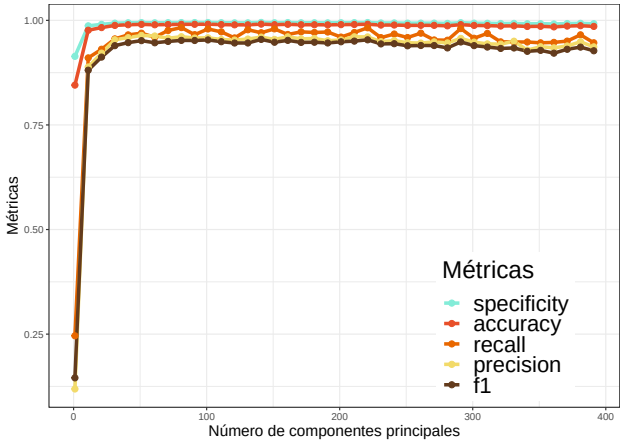

Figura 9: Número de componentes principales frente principales métricas

Los resultados reafirman la hipótesis que lanzamos al aire en el principio de esta subsección: con menos de 100 componentes principales, podemos explicar la gran mayoría de los resultados. Concretamente optaremos por 31 componentes principales, porque es el primer subconjunto de componentes que nos otorga (en el **conjunto de test**) una *accuracy* del 99% (aún no siendo esta una buena métrica de selección de modelo, todas presentan sus primeros valores "altos" entorno a este número de componentes, por lo que se menciona la selección por mera

conveniencia).

Es importante recalcar que los resultados obtenidos con PCA no son definitivos, sino que se han creado como orientación para acotar el número de componentes principales óptimas: el modelo que se ha entrenado con cada uno de los subconjuntos puede ser mejorado aún más si se analiza con profundidad el problema generado. Por ejemplo con la selección de 31 componentes principales, la topología que se ha utilizado en esta parte ha sido de un 90% de las capas de entrada, es decir, una capa oculta con 28 neuronas; no obstante, puede que esta configuración no sea la óptima (cosa que se comenta en la sección 6), pero sí nos sirve para acotar el número de componentes principales, ya que al fin y al cabo todas están siendo evaluadas bajo los mismos criterios.

Con este preprocesado de los datos, hemos logrado:

1. Reducir notoriamente el número de valores presentes en la base de datos (véase tabla 1).

	Número de atributos	Número de valores
Modelo completo	784	32.928.000
Modelo con 30 componentes principales	30	1.260.000
	Reducción del	96.17%

Tabla 1: Impacto en tamaño de aplicar PCA

2. Reducir los tiempos de entrenamiento de forma drástica, de tal modo, que aún siendo los resultados peores (que no son), tendría sentido optar por el peor modelo, pero con tiempos de ejecución "razonables".
   Véase en la figura 10, como el simple hecho de entrenar un modelo con aproximadamente 350 atributos nos sitúa en un tiempo de entrenamiento que ronda la hora, frente a uno de 30 atributos que se sitúa entre 5 y 10 minutos. Recuérdese que originalmente nuestra base de datos contaba con 784 atributos.
3. No sólo se ahorra tiempo y se reduce el tamaño de la base de datos, sino que como consecuencia de ambas se reduce el tráfico por RAM: en los primeros 150 modelos entrenados se transfirieron por RAM 30GiB aproximadamente, mientras que en los últimos 100 (subconjuntos de componentes principales de 300 a 400 de 10 en 10), un total de 11TiB.

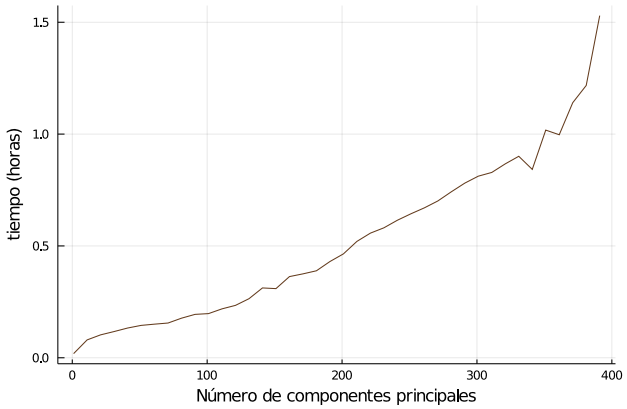

Figura 10: Tiempo de entrenamiento por cada subconjunto de componentes principales

6. Desarrollo

Durante los siguientes subapartados trataremos diferentes aproximaciones para resolver el problema propuesto: mediante el uso de redes neuronales artificiales (RR.NN.AA.) con la base de datos en primer lugar y con las proyecciones generadas con PCA en el segundo caso.

6.1. RR.NN.AA.

Como primera aproximación entrenamos una red neuronal artificial, utilizando siempre como función de de transferencia la sigmoidal 1 utilizando tres topologías diferentes.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Entrenaremos cada RNA como un problema multiclase, ya que las RR.NN.AA. nos permiten hacerlo de forma nativa, sin la necesidad de implementar una estrategia uno contra todos o uno contra uno. Para ello, simplemente tendremos que disponer de tantas neuronas como clases en la capa de salida y aplicaremos la función *softmax* para que la suma de valores de

la capa de salida (la probabilidad de pertenencia a cada clase) sea igual a 1.

6.1.1. Topología con 800 neuronas en una capa oculta

En este primer intento trabajaremos con 784 neuronas en la capa de entrada (tantas como píxeles tiene nuestra imagen, que recordemos es de dimensión 28×28. Contaremos con una capa oculta de 800 neuronas ($H_1 \dots H_{800}$ y una capa de salida con 10 ($c_1 \dots c_{10}$) como se puede ver en la figura 11.

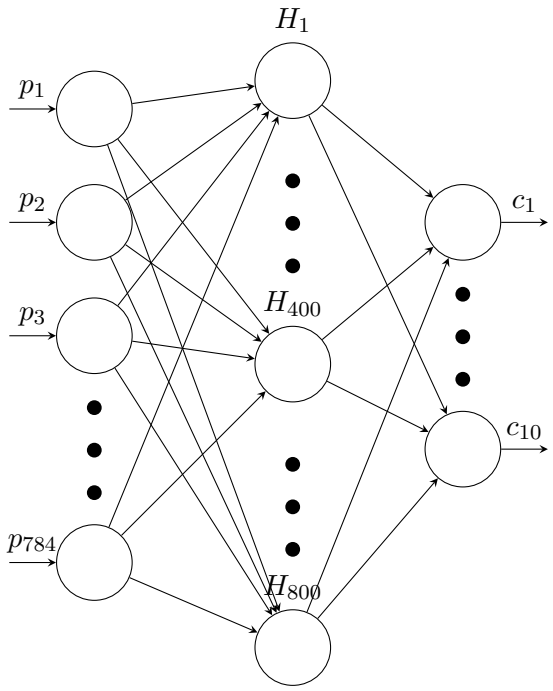

Figura 11: Representación gráfica de la topología de 800 neuronas.

Hemos escogido esta topología como punto de partida basándonos en el trabajo de Simard et al., 2003.

Recordamos, como se mencionó en la sección de preprocesado (5), los datos de entrada han sido (y deberán ser) normalizados para evaluar la red.

6.1.2. Topología con 600 neuronas en una capa oculta

Para este segundo caso, contaremos con el mismo número de neuronas en las capas de entrada y salida, pero cambiaremos la cantidad de neuronas en la capa oculta a 600, como "pequeña" modificación de la descrita en el subapartado anterior.

6.1.3. Topología con 2 capas ocultas

Finalmente, plantearemos un último modelo, el cual dispondrá de dos capas ocultas, con 300 y 100 neuronas respectivamente (figura 12). Probaremos esta configuración, que es la que se utiliza al trabajar con una base de datos similar, la de MNIST Fashion en Géron, 2019.

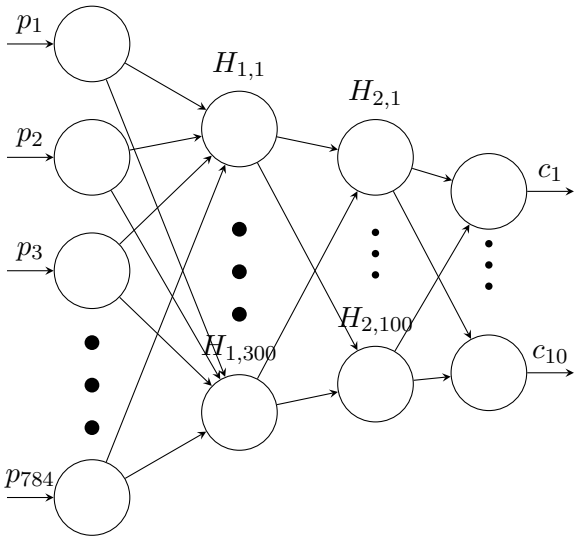

Figura 12: Red neuronal artificial densa con dos capas ocultas.

Resultados

Las figuras 13, 14 y 15 representan los errores de los conjuntos de entrenamiento (naranja), validación (celeste) y test (marrón) durante el proceso de entrenamiento y los resultados que aporta son cuanto menos, interesantes.

Las tres topologías terminan llegando a errores *crossentropy* similares, pero el número de iteraciones (*epochs*) que les lleva es sumamente distinto: en la gráfica 16 (un *violin plot*, similar a un diagrama de cajas), podemos observar la clara diferencia en el número de iteraciones, siendo la topología con 2 capas ocultas la más lenta (casi el doble de iteraciones por entrenamiento), frente a las monocapa, donde curiosamente, la red con la capa de 800 neuronas es ligeramente más rápida que la de 600.

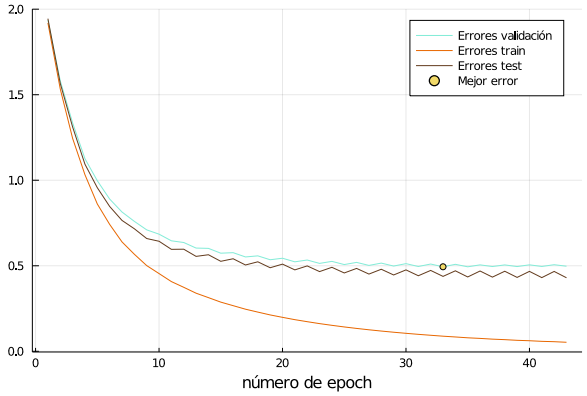

Figura 13: Errores *crossentropy* en la red con 800 neuronas en la capa oculta

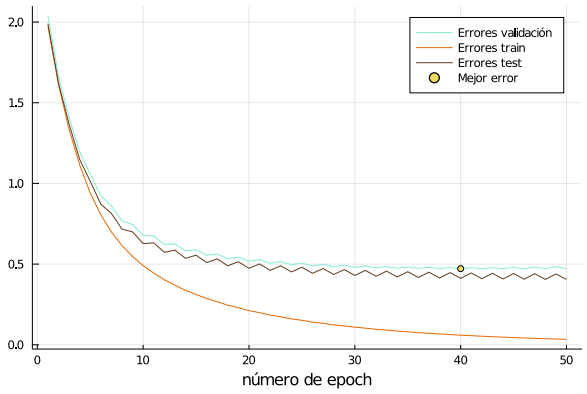

Figura 14: Errores *crossentropy* en la red con 600 neuronas en la capa oculta

Validación Cruzada

Como procedimiento para comprobar la robustez de los modelos propuestos, y para seleccionar uno en función de sus métricas, utilizaremos el método de *K-Fold Crossvalidation*, particionando nuestro conjunto de datos en 10 *folds* y entrenando cada vez con 9 de ellos y reservando 1 para realizar test y calcular las métricas.

Como las RR.NN.AA. no son modelos deterministas, para cada *fold* ejecutaremos el proceso 5 veces, esto es, creando y entrenando una red de neuronas 5 veces y promediaremos los resultados de estas ejecuciones.

Tras aplicar validación cruzada obtenemos los siguientes resultados para cada topología propuesta. En cada tabla se muestran en columnas las métricas, en las 10 primeras filas los

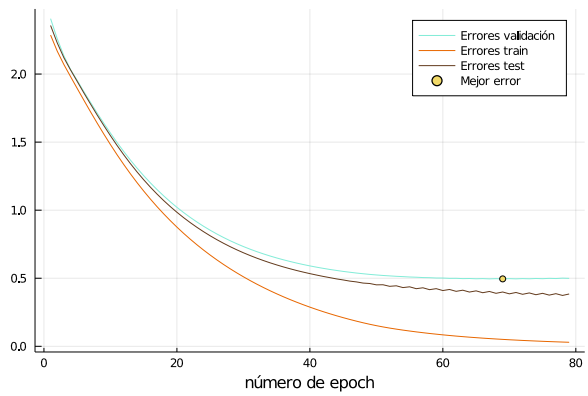

Figura 15: Errores *crossentropy* en la red con 2 capas ocultas de 300 y 100

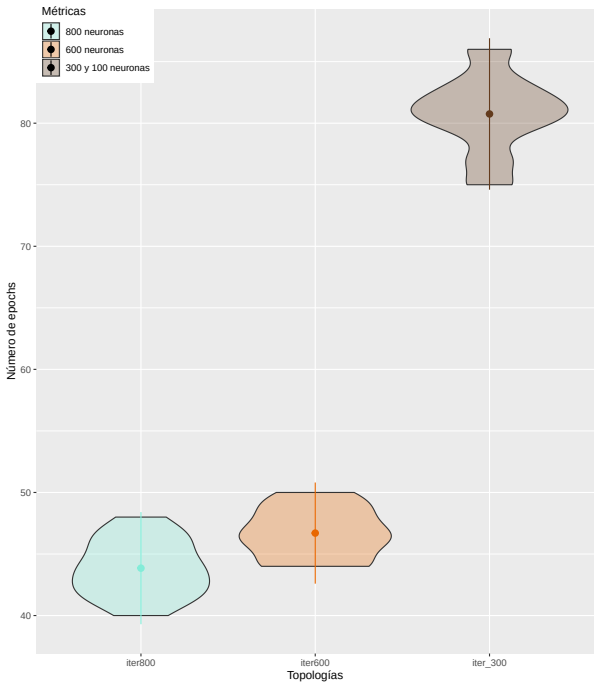

Figura 16: Representación en distribución de las epochs de las topologías ([800], [600], [300,100])

folds, en μ la media y en σ la desviación típica de las métricas obtenidas.

Como podemos observar en las tres tablas (tablas 2, 3 y 4), hemos obtenido resultados muy buenos en las topologías escogidas. Todas cuentan con altas medias para cada métrica y con una desviación típica muy baja, lo cual indica que las métricas suelen tomar valores muy similares, traduciéndose en que no estamos sobreentrenando y que los modelos se

generalizan bien a la hora de hacer predicciones con nuevos datos.

	Accuracy	Recall	Precision	F1 Score	Specificity	NPV
1	0.9792	0.8948	0.8958	0.8948	0.9884	0.9886
2	0.9786	0.8914	0.8932	0.8912	0.9880	0.9880
3	0.9796	0.8958	0.8972	0.8958	0.9884	0.9886
4	0.9794	0.8956	0.8970	0.8954	0.9886	0.9886
5	0.9792	0.8946	0.8962	0.8946	0.9884	0.9884
6	0.9788	0.8930	0.8946	0.8928	0.9882	0.9882
7	0.9782	0.8898	0.8920	0.8900	0.9878	0.9878
8	0.9790	0.8924	0.8938	0.8922	0.9880	0.9880
9	0.9788	0.8930	0.8950	0.8934	0.9886	0.9886
10	0.9790	0.8944	0.8956	0.8942	0.9882	0.9882
μ	0.9790	0.8930	0.8950	0.8930	0.9880	0.9880
σ	10^{-3}	$3 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$4 \cdot 10^{-4}$	10^{-3}

Tabla 2: Métricas para una capa oculta con 800 neuronas.

	Accuracy	Recall	Precision	F1 Score	Specificity	NPV
1	0.9796	0.8956	0.8966	0.8956	0.9886	0.9886
1	0.9794	0.8952	0.8970	0.8954	0.9886	0.9886
3	0.9792	0.8956	0.8966	0.8954	0.9886	0.9886
4	0.9798	0.8968	0.8982	0.8970	0.9888	0.9888
5	0.9798	0.8982	0.8996	0.8984	0.9890	0.9890
6	0.9794	0.8974	0.8982	0.8974	0.9890	0.9890
7	0.9796	0.8964	0.8980	0.8964	0.9886	0.9886
8	0.9794	0.8962	0.8976	0.8962	0.9886	0.9890
9	0.9798	0.8974	0.8984	0.8974	0.9888	0.9888
10	0.9796	0.8960	0.8974	0.8960	0.9888	0.9888
μ	0.9800	0.8960	0.8980	0.8970	0.9890	0.9890
σ	$5 \cdot 10^{-4}$	$2.4 \cdot 10^{-3}$	$2.1 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$

Tabla 3: Métricas para una capa oculta con 600 neuronas.

	Accuracy	Recall	Precision	F1 Score	Specificity	NPV
1	0.9804	0.9008	0.9016	0.9008	0.989	0.9890
2	0.9800	0.8994	0.9000	0.8994	0.989	0.9890
3	0.9800	0.8988	0.8998	0.8990	0.989	0.9890
4	0.9802	0.8990	0.9002	0.8994	0.989	0.9890
5	0.9802	0.9010	0.9018	0.9010	0.989	0.9890
6	0.9804	0.9004	0.9010	0.9004	0.989	0.9890
7	0.9804	0.9002	0.9008	0.9002	0.989	0.9890
8	0.9804	0.9014	0.9018	0.9012	0.989	0.9890
9	0.9802	0.9004	0.9008	0.9004	0.989	0.9890
10	0.9804	0.8992	0.9000	0.8992	0.989	0.9892
μ	0.9800	0.9000	0.9010	0.9000	0.989	0.9890
σ	$3 \cdot 10^{-4}$	$1.8 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$2 \cdot 10^{-3}$

Tabla 4: Métricas para dos capas ocultas, de 300 y 100 neuronas.

Tras ver estos resultados, podemos concluir que el mejor modelo parece ser que es el propuesto en Géron, 2019, con dos capas ocultas, de 300 y 100 neuronas.

Discusión

Los resultados obtenidos, como se acaba de comentar, han sido *espectaculares*, pero con el costo que tiene construir la validación cruzada con este número de parámetros y observaciones: dicho entrenamiento le llevó entorno a las 21 horas y generó un tráfico de 32TiB en RAM, utilizando para ello un procesador Intel Core i9 de 10 núcleos a 3.6GHz con 32GiB de RAM.

Como se ha visto en el análisis exploratorio de los datos, concretamente en el mapa de calor (figura 3), gran parte de la información se concentra en una zona concreta, por lo que podríamos reducir la zona que utilizamos para entrenar, sin para ello perder información o calidad en los modelos. Esto, juntado con los tiempos de ejecución que obtenemos entrenando con toda la base de datos, nos da pie a utilizar las proyecciones que hemos introducido en la sección 5.

6.2. RR.NN.AA. usando PCA

En esta segunda aproximación, observaremos los resultados de sacrificar variabilidad de las explicativas originales a cambio de beneficiarse de la velocidad y simplicidad de entrenar con modelos mucho más pequeños.

En esta ocasión, los parámetros de entrada han cambiado; obsérvese cómo se ven afectadas el número de neuronas en esta nueva configuración tal y como se explicó en la sección 5: en la primera aproximación contábamos con 784 neuronas de entrada (véase imagen 11), frente a las 31 de este nuevo punto de vista (figura 17). Se ha utilizado, al igual que en la anterior aproximación, varias topologías para probar esta nueva forma de los datos, todas de una capa oculta, de 35, 28 y 15 neuronas.

De esta forma, aunque sea beneficioso a la hora de medir tiempos, debemos tener en cuenta de que, cada vez que queramos evaluar una observación deberemos de "pasarla" a esta nueva escala; para ello, sea M la matriz de proyección, de tamaño 784x31, y una nueva imagen img de tamaño 1x784 (es decir, una imagen con 784 píxeles), el input de la red vendrá dado por:

$$\text{proyección} = img \cdot M \tag{2}$$

Para el proceso de entrenamiento hemos usado

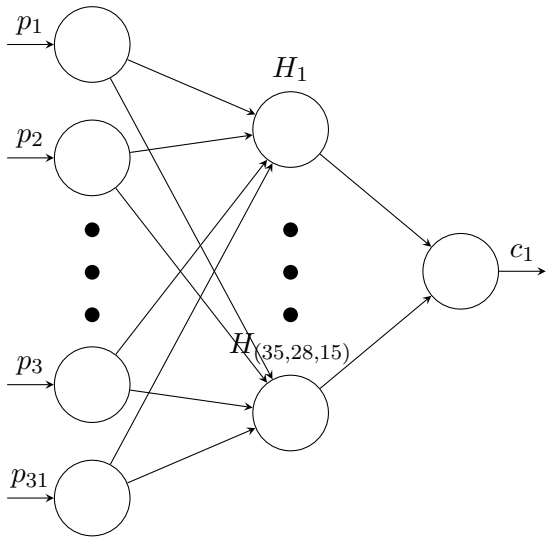

Figura 17: Red neuronal artificial densa con el número de entradas reducido.

un *ADAM* (tasa de aprendizaje) de 0.005 en todas las topologías, se ha utilizado un número máximo de iteraciones de 500 y como criterio de parada temprana que detuviera el entrenamiento en caso de haber estado 20 iteraciones sin mejorar el error en test. Además, como se mencionó en la sección de preprocesado (5), se han normalizado los datos, después se han calculado las proyecciones, y ahora hemos vuelto a normalizar los resultados de las proyecciones[2].

6.2.1. Topología con 35 neuronas en una capa oculta

Como se ha mencionado, la capa de entrada estará formada por 31 neuronas (las correspondientes a las proyecciones de las primeras 31 componentes principales) y tiene en su capa oculta 35 neuronas. Esta configuración se ha escogido puesto que ha sido la topología con el mayor número de neuronas ocultas con resultados "decentes".

6.2.2. Topología con 28 neuronas en una capa oculta

Misma configuración que la topología de 35 neuronas en la capa oculta, pero con 28. La elección de este modelo viene dada a que es la correspondiente configuración que se utilizó para la selección de las configuraciones al realizar PCA ($\lceil 90\% \cdot 31 = 28 \rceil$); por lo que puede ser interesante comprobar cuánto se ha visto beneficiado/perjudicado este modelo en concreto.

6.2.3. Topología con 15 neuronas en una capa oculta

Igual que las dos anteriores, pero con 15 neuronas en la capa oculta. Se ha seleccionado con el mismo criterio que la de 35: fue el número de neuronas más bajo que obtuvo buenos resultados.

Resultados

Las figuras 18, 19 y 20 representan los errores en los conjuntos de entrenamiento (naranja), validación (marrón) y test (celeste) durante el proceso de entrenamiento.

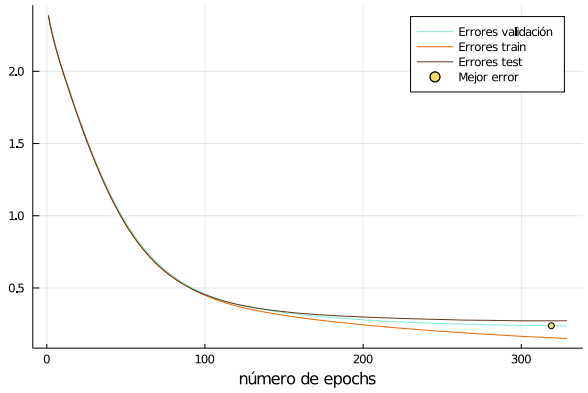

Figura 18: Errores *crossentropy* en la red con 35 neuronas en la capa oculta

Como curiosidad, a nivel de error de entropía cruzada, en este modelo obtenemos incluso mejores resultados que los propuestos con el modelo completo (véanse figuras 13, 14 y 15).

Esta mejora, no tiene por que traducirse en una mejora de las métricas, pero en esta ocasión, implica una ligera, pero existente, mejora de cada una de las métricas (se detallará más en el apartado de discusión de los resultados).

Los resultados (por validación cruzada) que hemos obtenido han sido los que se muestran en las tablas 5, 6 y 7.

Como se puede observar, no hay grandes

[2]Desconocemos el impacto que esto puede tener sobre los datos; hallamos tanto opiniones de que no es una buena praxis como que no afecta. De todos modos, hemos optado por realizar la normalización ya que nos proporcionó mejores resultados.

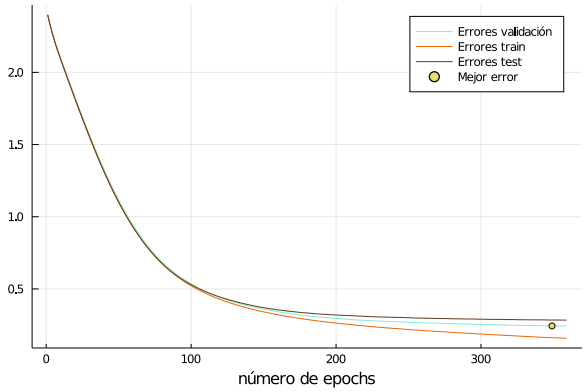

Figura 19: Errores *crossentropy* en la red con 28 neuronas en la capa oculta

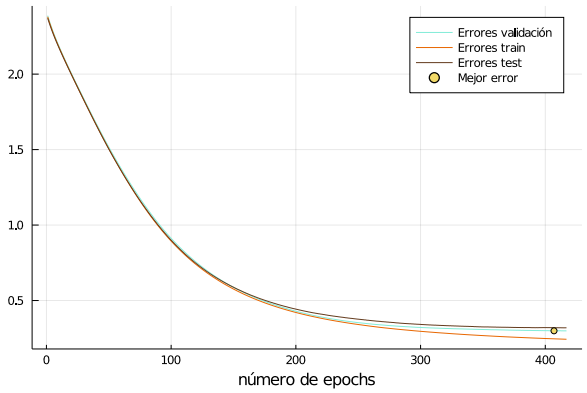

Figura 20: Errores *crossentropy* en la red con 15 neuronas en la capa oculta

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.9790	0.8928	0.8924	0.8926	0.9880	0.9880
2	0.9784	0.8914	0.8914	0.8914	0.9880	0.9880
3	0.9784	0.8920	0.8918	0.8916	0.9880	0.9880
4	0.9784	0.8914	0.8912	0.8912	0.9880	0.9880
5	0.9786	0.8912	0.8912	0.8912	0.9882	0.9882
6	0.9786	0.8928	0.8924	0.8922	0.9882	0.9882
7	0.9784	0.8916	0.8916	0.8916	0.9880	0.9880
8	0.9786	0.8920	0.8918	0.8918	0.9880	0.9880
9	0.9788	0.8920	0.8920	0.8920	0.9882	0.9882
10	0.9788	0.8934	0.8932	0.8932	0.9880	0.9880
μ	0.97860	0.89200	0.89190	0.89180	0.98810	0.98810
σ	$5 \cdot 10^{-4}$	$2.3 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$

Tabla 5: Métricas para la capa oculta de 35 neuronas.

diferencias entre las tres configuraciones. Teniendo en cuenta el preprocesado (5) que llevan detrás, sobre todo la reducción de dimensionalidad, y como consecuencia de tamaño de la base de datos, los resultados son increíblemente buenos (recuérdese que todos los

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.97664	0.88142	0.88200	0.88146	0.98702	0.98702
2	0.97686	0.88250	0.88302	0.88256	0.98714	0.98716
3	0.97676	0.88202	0.88254	0.88202	0.98710	0.98714
4	0.97690	0.88268	0.88296	0.88262	0.98718	0.98718
5	0.97696	0.88298	0.88330	0.88294	0.98722	0.98724
6	0.97634	0.87998	0.88026	0.87994	0.98686	0.98688
7	0.97644	0.88062	0.88106	0.88054	0.98692	0.98694
8	0.97658	0.88126	0.88186	0.88130	0.98700	0.98704
9	0.97688	0.88268	0.88292	0.88260	0.98714	0.98718
10	0.97644	0.88058	0.88102	0.88054	0.98692	0.98692
μ	0.97670	0.88170	0.88210	0.88170	0.98700	0.98710
σ	$6 \cdot 10^{-4}$	$2.8 \cdot 10^{-3}$	$2.8 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$3 \cdot 10^{-3}$

Tabla 6: Métricas para la capa oculta de 28 neuronas.

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.9764	0.8814	0.8824	0.8814	0.9870	0.9870
2	0.9768	0.8826	0.8828	0.8826	0.9870	0.9870
3	0.9768	0.8824	0.8826	0.8822	0.9870	0.9870
4	0.9770	0.8828	0.8830	0.8826	0.9870	0.9870
5	0.9770	0.8832	0.8834	0.8830	0.9872	0.9872
6	0.9764	0.8800	0.8804	0.8800	0.9868	0.9870
7	0.9764	0.8806	0.8810	0.8806	0.9870	0.9870
8	0.9766	0.8814	0.8820	0.8814	0.9870	0.9870
9	0.9770	0.8828	0.8830	0.8828	0.9870	0.9870
10	0.9762	0.8804	0.8812	0.8804	0.9870	0.9870
μ	0.9770	0.8820	0.8820	0.8820	0.9870	0.9870
σ	$6 \cdot 10^{-4}$	$2.8 \cdot 10^{-3}$	$2.8 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$

Tabla 7: Métricas para la capa oculta de 15 neuronas.

resultados que se muestran son en el conjunto de test).

Aun siendo los resultados tan buenos, hemos tratado de mejorarlos: en principio, el hecho de entrenar teniendo 10 neuronas de salida, debería ser similar a entrenar en "uno contra todos"; no obstante, decidimos comprobarlo. En las siguientes tablas (8, 9 y 10) se observan las métricas obtenidas al entrenar en validación cruzada en uno contra todos (interesante que este proceso tomó entorno a 3-4h -en uno contra todos, es decir, 10 modelos por entrenamiento-, frente a las 21 del modelo completo).

Curiosamente, los resultados no son muy distintos, pero si observamos diferencias: de media, el hecho de entrenar en uno contra todos, genera una mejora de un 1% en cada una de las métricas, lo cual no es una mejora claramente significativa, pero si implica una mejora (al coste de entrenar 10 modelos en vez de 1).

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.983876	0.918877	0.919399	0.919138	0.991040	0.991057
1	0.984314	0.921309	0.922071	0.921689	0.991288	0.991307
3	0.982829	0.912447	0.913981	0.913213	0.990456	0.990489
4	0.981571	0.906580	0.908251	0.907414	0.989772	0.989813
5	0.983200	0.914469	0.915226	0.914847	0.990675	0.990700
6	0.981971	0.907272	0.908787	0.908028	0.989994	0.990033
7	0.983305	0.915990	0.916678	0.916334	0.990729	0.990754
8	0.981743	0.907291	0.909164	0.908226	0.989860	0.989909
9	0.981667	0.907879	0.907876	0.907878	0.989822	0.989855
10	0.980486	0.902438	0.902889	0.902663	0.989160	0.989195
μ	0.982000	0.911000	0.912000	0.912000	0.990000	0.990000
σ	10^{-3}	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	10^{-3}	10^{-3}

Tabla 8: Métricas para la capa oculta de 35 neuronas en uno contra todos.

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.983876	0.918877	0.919399	0.919138	0.991040	0.991057
2	0.984314	0.921309	0.922071	0.921689	0.991288	0.991307
3	0.982829	0.912447	0.913981	0.913213	0.990456	0.990489
4	0.981571	0.906580	0.908251	0.907414	0.989772	0.989813
5	0.983200	0.914469	0.915226	0.914847	0.990675	0.990700
6	0.981971	0.907272	0.908787	0.908028	0.989994	0.990033
7	0.983305	0.915990	0.916678	0.916334	0.990729	0.990754
8	0.981743	0.907291	0.909164	0.908226	0.989860	0.989909
9	0.981667	0.907879	0.907876	0.907878	0.989822	0.989855
10	0.980486	0.902438	0.902889	0.902663	0.989160	0.989195
μ	0.982000	0.911000	0.912000	0.912000	0.990000	0.990000
σ	10^{-3}	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	$6 \cdot 10^{-3}$

Tabla 9: Métricas para la capa oculta de 28 neuronas en uno contra todos.

	Accuracy	Recall	Precision	F_1 Score	Specificity	NPV
1	0.981943	0.908828	0.909847	0.909336	0.989968	0.990002
2	0.982610	0.912215	0.912685	0.912450	0.990342	0.990367
3	0.982724	0.912603	0.913475	0.913038	0.990402	0.990431
4	0.982476	0.911832	0.912873	0.912352	0.990265	0.990287
5	0.982648	0.913444	0.913390	0.913417	0.990354	0.990373
6	0.982419	0.912003	0.912229	0.912116	0.990228	0.990244
7	0.983181	0.913669	0.914509	0.914089	0.990672	0.990689
8	0.981686	0.906746	0.906850	0.906798	0.989838	0.989858
9	0.982019	0.908869	0.909945	0.909406	0.990013	0.990049
10	0.982210	0.910875	0.911018	0.910947	0.990117	0.990140
μ	0.982000	0.911000	0.912000	0.911000	0.990000	0.990000
σ	$4.1 \cdot 10^{-4}$	$2.14 \cdot 10^{-3}$	$2.17 \cdot 10^{-3}$	$2.15 \cdot 10^{-3}$	$2.3 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$

Tabla 10: Métricas para la capa oculta de 15 neuronas en uno contra todos.

Discusión

No consideramos que la mejora de entrenar en uno contra todos frente a un problema multiclase genere una mejora tan significativa como para que compense el coste de tener que entrenar 10 modelos en vez de 1 (sobre todo por el tiempo que esto supone).

No obstante, curiosamente, al entrenar en uno contra todos, obtenemos mejores resultados (a nivel de métricas) que los obtenidos entrenando con la base de datos completa. Esto puede deberse a la selección de topologías no apropiadas, pero para su elección, no fuimos capaces de superar las métricas que las obtenidas con el modelo propuesto por Géron, 2019.

7. Conclusiones

Tras analizar los resultados obtenidos, podemos concluir que aplicar PCA puede ser una muy buena práctica; ya que obtenemos resultados muy buenos, reduciendo los tiempos de entrenamiento y la complejidad del problema.

Tanto por los resultados, como por la simplicidad, creemos firmemente que el mejor modelo reside en cualquiera (todos son buenos) de los que hemos aplicado PCA. La mayor complejidad del problema reside en tratar las 42000 observaciones con 784 variables, por capacidades computaciones, por lo que el hecho de obtener resultados tan buenos en PCA es un muy buen logro.

8. Trabajo futuro

Una forma de mejorar aún más los resultados podría pasar por realizar un preprocesado más profundo, aplicando técnicas de *blur*, *thinning*, otras como hace LeCun et al., 1990 introduciendo redes convolucionales como parte del preprocesado o incluso aplicando una reducción del tamaño de las imágenes: posiblemente podría reducirse la imagen (quizás con una imagen 16x16 tengamos información suficiente) o probar a aplicar técnicas diferentes de reducción de la dimensionalidad como podrían ser LDA (*Linead Discriminant Analysis*), *Isompap*, MDS (*Multidimensional Scaling*) o incluso intentar utilizar la técnica de visualización *t-SNE* mencionada en secciones anteriores.

Puede ser interesante tratar de afrontar este problema con técnicas de *machine learning* diferentes como SVM (*Support Vector Machine*), árboles de decisión y sobre todo, mediante redes convolucionales. Además, este trabajo puede ser tomado como punto de partida en un modelo más grande que trate de clasificar todos os caracteres de la base de datos completa del NIST.

Tabla de contenidos

Referencias

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11).

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 396–404.

Hinton, G., & Roweis, S. T. (2002). Stochastic neighbor embedding. *NIPS*, *15*, 833–840.

Simard, P. Y., Steinkraus, D., Platt, J. C., et al. (2003). Best practices for convolutional neural networks applied to visual document analysis. *Icdar*, *3*(2003).

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(11).

Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2011). Convolutional neural network committees for handwritten character classification. *2011 International Conference on Document Analysis and Recognition*, 1135–1139.

Babu, U. R., Venkateswarlu, Y., & Chintha, A. K. (2014). Handwritten digit recognition using k-nearest neighbour classifier. *2014 World Congress on Computing and Communication Technologies*, 60–65. https://doi.org/10.1109/WCCCT.2014.7

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.