UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
MAESTRÍA EN CIENCIAS MATEMÁTICAS

Proyecto Final: Redes Neuronales (NN)

Alarcón-González, Cerezo-Silva, Zarco-Romero

Curso: *Aprendizaje Estadístico Automatizado* – Profesora: *Dra. Guillermina Eslava Gómez*

22 de junio de 2021

DNN

Para esta sección se utilizó la librería `h2o` con la cual se generaron 3 modelos simples diferentes de Redes Neuronales Profundas (por sus siglas en inglés *DNN*), en particular se utilizó la función `h2o.deeplearning()`, con la finalidad de explorar el poder predictivo del algoritmo variando algunos de los parámetros de tal manera que el modelo no tardara demasiado tiempo en correr y al mismo tiempo se lograran explorar las tasas de error en los modelos. Los resultados de este proceso se ven en la Tabla 1, donde se observa que **el modelo 2 es el que tiene un menor error de entrenamiento y de prueba** que los otros modelos, por lo que es el mejor candidato a modelo predictivo entre los propuestos. Una observación interesante es que el modelo 1 a pesar de tener un error de entrenamiento menor que el modelo 3, el error de prueba resulta mayor en el modelo 1 que el del modelo 3, algo que da indicios de un sobreajuste ligero por parte del modelo 1.

Model	RunTimeMins	Layers	Epochs	%Loglss_trn	%Loglss_tst	%Err_trn	%Err_tst
2	9.86	500-500-500-500-500	25	0.45	38.88	0.10	2.63
3	60.19	2500-2500	40	2.18	34.47	0.27	2.68
1	1.12	200-200	10	1.00	23.84	0.24	2.97

Tabla 1: Estadísticas y parámetros de los modelos simples ajustados. *Nota: Todos los parámetros que no estén aquí mencionados fueron tomados por default.*

A continuación, vamos a explorar la efectividad por clase ($\{0, 1, 2, \ldots, 9\}$) de cada uno de los modelos simples. En la Tabla 2 se presenta el porcentaje de error en el conjunto de entrenamiento en cada clase. Por otro lado, en la Tabla 3 se muestra lo análogo para el conjunto de prueba. En la Figura 2 podemos ver el gráfico de las anteriores tablas, aquí apreciamos que no hay un modelo que domine completamente para todas las clases, aunque parece que la clase más sencilla de asignar al menos de forma general es la correspondiente al número 0.

Model	0	1	2	3	4	5	6	7	8	9	Total
2	0.00	0.02	0.05	0.22	0.31	0.08	0.03	0.14	0.00	0.15	0.10
3	0.46	0.02	0.33	0.32	0.13	0.49	0.08	0.26	0.52	0.15	0.27
1	0.13	0.18	0.03	0.78	0.46	0.11	0.00	0.07	0.36	0.25	0.24

Tabla 2: **Porcentaje de error** de los modelos simples por clase y total en el conjunto de **entrenamiento**.

En la Figura 3 se muestran algunos de los valores ajustados, la respuesta original y el gráfico del número que fue trazado. En dicha figura se presentan 9 bien asignados y 9 mal asignados en forma de matriz de 3×3.

Todo esto da una idea de qué parámetros pueden ser apropiados para proponer un modelo. De tal manera que a continuación, usando la función `h2o.grid(algorithm="deeplearning")` y basados en los resultados expuestos anteriormente, procedemos a explorar una **malla** de modelos dada por una variación de parámetros que

Model	0	1	2	3	4	5	6	7	8	9	Total
2	1.12	1.47	2.86	4.56	3.32	2.86	1.55	2.27	3.28	3.18	2.63
3	1.35	1.28	2.42	4.77	1.72	2.98	1.66	2.38	5.20	3.18	2.68
1	1.12	1.77	2.53	7.42	2.63	2.73	1.88	2.16	4.64	2.71	2.97

Tabla 3: **Porcentaje de error** de los modelos simples por clase y total en el conjunto de **prueba**.

tengan características similares a las de los modelos anteriores que y que tiendan a la composición del mejor modelo simple. La idea para crear la malla paramétrica que propondremos será basarnos en un esquema aleatorio pero con una **tendencia** al modelo simple 2. Además de que se ha integrado la variación del parámetro `l1` con la finalidad de lidiar con el sobreajuste y también se realizó una calibración por validación cruzada con `nfolds` $= 4$. La manera que diseñamos la arquitectura para los hiperparámetros de la malla aleatoria será la siguiente:

- La cantidad de capas estará dada de forma aleatoria en el conjunto $\{2, 3, 4, 5, 6\}$ con probabilidades correspondientes $\{0.1, 0.25, 0.25, 0.3, 0.1\}$. Destacando la cantidad de capas del modelo simple 2.
- La cantidad de nodos en cada capa vendrá dada por aleatoriamente por el conjunto $\{200, 300, 500, 1000, 2500\}$ con probabilidades $\{0.25, 0.25, 0.3, 0.1, 0.1\}$ respectivamente. De nuevo, destacando la cantidad de nodos que se manejan en el modelo simple 2.
- Se crearán 100 capas ocultas construidas de forma aleatoria usando el esquema mencionado en los dos puntos anteriores.
- Para todas las 100 capas anteriores se seleccionará cada una de las épocas en el conjunto $\{10, 25, 40, 50\}$. Lo cual nos da 500 opciones diferentes hasta este punto.
- Para todas las 500 opciones anteriores se seleccionará cada uno de los parámetros `l1` en el conjunto de valores $\{0, 1e-5\}$. Lo cual nos da 1000 opciones diferentes hasta este punto.
- De estos 1000 escenarios, se seleccionarán aleatoriamente únicamente 20 modelos los cuales son los que conformarán la malla realizada.

Teniendo como base esta arquitectura, los resultados de los 10 mejores modelos tomando como referencia el error de prueba los podemos ver en la Tabla 4. Asimismo, en la Figura 4 tenemos el gráfico de los errores y la *logloss* contenidos tabla antes mencionada.

Model	RunTimeMins	layers	epochs	l1	%Loglss_trn	%Loglss_tst	%Err_trn	%Err_tst
8	9.21	1000-300-500	35.97	0	1.15	8.79	0.34	1.96
7	9.55	500-500-200-2500-300	40.21	0	2.47	12.22	0.74	2.39
18	5.62	500-500-200-300	38.12	0	1.99	10.30	0.66	2.46
4	3.98	300-500-500-500-300	25.33	0	4.36	12.44	1.35	2.79
12	6.79	2500-300	10.26	0	4.03	13.35	1.25	2.80
11	5.72	500-500-300-500	35.52	0	3.14	12.22	1.08	2.87
14	4.14	500-300-200-500	29.23	0	3.97	10.79	1.28	2.88
13	4.59	300-500-500-200	39.87	0	3.86	12.06	1.32	2.91
5	4.23	300-500-500-200	36.70	0	4.11	12.97	1.39	2.94
3	4.54	500-200-500-300	33.95	0	4.03	13.64	1.30	2.96

Tabla 4: Estadísticas y parámetros de los 10 mejores modelos de los 20 obtenidos en la malla ordenados por error de prueba. *Nota: Todos los parámetros que no estén aquí mencionados fueron tomados por default.*

Con base en la Tabla 4, **el mejor candidato a modelo predictivo dado por la malla será el modelo 8**. De hecho este será el mejor modelo propuesto, superando a los obtenidos en las mallas simples en términos del error de

prueba. De los 3 mejores modelos por error de prueba en la malla mostramos a continuación los porcentajes de error obtenidos por clase en el conjunto de entrenamiento (Tabla 5) y de prueba (Tabla 6). Estos resultados se pueden observar gráficamente en la Figura 5 donde se nota consistentemente que el modelo 8 es el mejor candidato.

Model	0	1	2	3	4	5	6	7	8	9	Total
8	0.05	0.13	0.30	0.29	0.26	0.55	0.23	0.29	0.55	0.82	0.34
7	0.18	0.51	0.58	1.15	0.66	0.93	0.51	0.89	1.04	0.94	0.74
18	0.20	0.58	0.40	1.02	0.82	0.74	0.25	0.65	1.40	0.60	0.66

Tabla 5: **Porcentaje de error** de los 3 mejores modelos de la malla por clase y total en el conjunto **training**.

Model	0	1	2	3	4	5	6	7	8	9	Total
8	0.79	0.49	2.09	3.82	1.60	2.48	1.44	1.52	3.05	2.47	1.96
7	0.67	0.98	2.53	4.88	2.06	2.61	1.88	2.49	3.73	2.12	2.39
18	1.01	1.18	2.09	4.88	2.41	2.11	1.99	1.62	5.09	2.24	2.46

Tabla 6: **Porcentaje de error** de los 3 mejores modelos de la malla por clase y total en el conjunto de **prueba**.

Por lo tanto, el modelo que se propondrá por parte del algoritmo *DNN* será el modelo 8. El cual produce las estimaciones dadas para el conjunto de validación que se guardó en el archivo `Proyecto_Alarcon_DNN_pred.csv` . Algunos de los resultados tanto bien como mal asignados, los podemos ver en la Figura 1.

Figura 1: Matrices de números bien y mal asignados en el conjunto de **prueba** de acuerdo al mejor modelo modelo de la malla *(DNN)*. En la matriz de 3×3 de la izquierda vemos los bien asignados y en la de la derecha los mal asignados.

Regresión logística

Dado que el número de categorías es mayor a dos, para ajustar un modelo regularizado se hizo una regresión multinomial, ocupando la función `cv.glmnet` y calibrando con diferentes valores de $\alpha \in [0,1]$. Recordando que $\alpha = 1$ ajusta la regresión LASSO y $\alpha = 0$ la regresión RIDGE.

Se realizaron $B = 3$ repeticiones para $\alpha \in \{0, 0.2, 0.5, 0.7\}$ aplicando validación cruzada con $nfolds = 10$.

Inicialmente se pretendió experimentar más, es decir, más repeticiones por modelo, más valores de α y diferentes particiones para la validación cruzada, sin embargo, se tuvo el problema del tiempo de ejecución de cada modelo (1 hr y 40 min en promedio) además de que para $\alpha > 0.7$ la función no converge en el número máximo de iteraciones (100, 000). Resulta interesante también el comentar que se buscó ocupar una instancia de Google con un costo de \$0.71 dólares la hora, con un procesador de 16 núcleos y memoria RAM de $64GB$, sin embargo, esto no mejoró los tiempos, pues la función `cv.glmnet` permite la ejecución en paralelo en solo una parte del proceso, en otras palabras, la estimación de un modelo tardó tan solo un poco menos (1 hr 20 min) dado que solo unos minutos se ocupan todos los núcleos y el resto del tiempo se ocupa sólo uno. Por tales motivos se descartó la posibilidad de contratar el servicio.

En la Tabla 7 se reportan lan median de las tres repeticiones, esto es, el valor de λ que minimiza los errores de prueba, para $\alpha \in \{0, 0.7\}$ se usó λ_{min} y para $\alpha \in \{0.2, 0.5\}$ se usó λ_{1se}, así como los errores porcentuales de entrenamiento y de prueba.

Model	α	λ	Non_zero	%Err_trn	%Err_tst
1	0.0	167.05	714	7.55	8.37
2	0.2	16.44	305	6.66	7.80
3	0.5	8.95	245	6.72	7.94
4	0.7	3.89	269	6.28	7.83

Tabla 7: Media de parámetros de ajuste, errores aparentes y de prueba de cuatro modelos multinomiales ajustados con la base `MNISTtrain` (B=3 repeticiones)

A continuación, vamos a explorar la media de efectividad por clase ($\{0, 1, 2, \ldots, 9\}$) de los distintos modelos multinomiales ya descritos. En la Tabla 8 se presenta la media del porcentaje de error en el conjunto de prueba en cada clase, de aquí podemos concluir que los dígitos que más error tienen son el 8, 3 y 5, y los que mejor predice son el 0, 1 y 6 .

Model	0	1	2	3	4	5	6	7	8	9	Total
1	2.13	2.55	10.99	13.26	5.96	13.54	6.31	7.03	13.24	9.78	8.37
2	2.02	2.78	10.22	12.65	5.77	12.71	5.91	6.46	13.01	9.42	7.80
3	2.06	2.68	10.33	12.87	5.92	12.55	6.13	6.39	13.27	9.74	7.94
4	2.02	3.14	9.78	12.41	5.73	11.55	5.54	6.28	13.12	9.66	7.83

Tabla 8: Media de los porcentajes de error de los modelos multinomiales por clase y total en el conjunto deprueba (B=3 repeticiones).

Con base en los resultados anteriores, se concluye que el mejor ajuste se tiene con $\alpha = 0.2$, por tal motivo es este el parámetro que se usó para predecir los valores de la base de validación.

En la Figura 6 vemos algunos de los valores ajustados, la respuesta original y el gráfico del número que fue trazado. En dicha figura se presentan 9 bien asignados y 9 mal asignados en forma de matriz de 3×3.

Random Forest

Experimentamos primero con `h2o` . En este caso, `h2o` trabajó con un máximo de 714 predictores, puesto que los otros 70 predictores eran columnas con datos constantes. Con dicho software, fijamos el parámetro `ntrees=500` y calibramos `mtries`. Además, recordemos que en el caso de clasificación, la teoría (Capítulo 5 de *An Introduction to Statistical Learning*) sugiere utilizar `mtries=`$\sqrt{784} = 28$. De este modo, ajustamos 5 modelos con `mtries = 26, 28, 30, 100, 714`. En el caso `mtries=100` tuvimos que cambiar el parámetro `max_depth=20` (valor default) por el parámetro `max_depth = 10`. El parámetro con los errores más bajos fue `mtries=28`.

Como segunda calibración, seguimos fijando `ntrees=500` y también `mtries=28`. Ajustamos dos modelos más con los valores `max_depth`=10,21. El modelo con el error global más bajo fue el ajustado con `max_depth=21`.

Posteriormente incrementamos el número de árboles, consideramos dos modelos más con `ntrees=800, 1000`. Sin embargo, con `ntrees=1000`, `h2o.randomforest` no terminó de ajustar, indicó un ajuste máximo del modelo del 96 %. Por ello, optamos por cambiar a la librería `randomForest`.

El resumen de lo anterior es el siguiente:

(i) `mtries`. Con `h2o` se ajustaron 5 modelos con `mtries` $= 26, 28, 30, 100, 714$ (Tabla 16 y Tabla 17 en anexo).

(ii) `max_depth`. Con `h2o` se utilizaron los valores `max_depth` $= 10, 20, 21$ (Tabla 18 y Tabla 19 en anexo).

(iii) `ntrees`. Con `h2o` se ajustó un modelo con `ntrees` $= 500$ y con `randomForest` dos modelos más con `ntrees` $= 800, 1000$ (Tabla 9 y Tabla 10).

(iv) `nodesize`. Con `randomforest` se utilizó el valor por default en todos los modelos (`nodesize=1`) y para el modelo final se incrementó a `nodesize=2`, pero en ese caso el error global aumentó. Por ello, se escogió el valor por default `nodesize=1`.

Las comparaciones de los errores de prueba y entrenamiento de los tres mejores modelos con Random Forest se encuentran en las tablas 9 y 10.

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`ntrees=500`	1.35	1.51	3.25	4.98	2.79	4.36	1.83	4.00	4.91	4.97	3.37
`ntrees=800`	1.35	1.54	3.45	4.73	2.99	3.98	1.78	3.88	4.81	5.24	3.36
`ntrees=1000`	1.32	1.40	3.50	4.76	2.86	4.09	1.81	3.78	4.81	5.22	3.36

Tabla 9: Errores OOB, en porcentaje, al variar `ntrees` con `randomforest()`, `mtry=28`.

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`ntrees=500`	0.67	1.96	3.41	6.79	2.98	3.73	3.32	3.57	4.30	3.53	3.42
`ntrees=800`	0.79	1.47	2.64	6.47	2.98	3.85	2.21	3.57	4.64	3.65	3.21
`ntrees=1000`	0.79	1.67	2.64	5.94	2.98	3.73	2.33	3.35	4.19	4.00	3.14

Tabla 10: *Test errors*, en porcentaje, al variar `ntrees` con `randomforest()`, `mtry=28`.

El modelo que minimiza el error de prueba es el que se calibra con 1000 árboles y tiene un parámetro de `mtry=28`, por ello se elige como modelo final. Los errores globales del modelo final se encuentran en la Tabla 13, mientras que los errores por clase del modelo final están el la Tabla 12.

	OOB	Test error	CV
Modelo final	3.36	3.14	3.65

Tabla 11: Errores globales, en porcentaje, del modelo final con randomForest(). Los parámetros del modelo son `ntree=1000` y `mtry=28`. El *test-error* se calculó para la base de datos MNISTtest. Para calcular el error por *cross-validation* se utilizaron $k = 3$-*folds*.

Errores \ *Clase*	0	1	2	3	4	5	6	7	8	9
OOB	1.32	1.40	3.50	4.76	2.86	4.09	1.81	3.78	4.81	5.22
Test	0.79	1.67	2.64	5.94	2.98	3.73	2.33	3.35	4.19	4.00

Tabla 12: Errores por clases, en porcentaje, del modelo final con `randomForest()` , con parámetros `ntree=1000` y `mtry=28`.

Comentarios.

- A diferencia de `h2o` , el software `randomForest` presenta los errores OOB en lugar de los errores de entrenamiento.
- Cada modelo se ajustó en un tiempo máximo de tres horas.
- El software `randomForest` fue más eficiente que `h2o` . Por ejemplo, `randomforest` admite un número mayor de árboles en tanto que `h2o` no concluía el ajuste del modelo con 800 árboles.
- El *test error* de la clase 3 es el más alto entre las 10 clases bajo el modelo final con `randomForest()` (Tabla 12). El segundo *test error* más alto es el de la Clase 8. Esto puede deberse a que confunde el número 3 con el número 8 y vicersa.
- La clase 0 tiene el *test error más bajo*, seguido de la clase 1 (Tabla 12).
- Un ajuste con `randomForest` y 500 árboles minimiza el error de prueba de la clase 0 en comparación con 800 y 1000 árboles. Sin embargo, con 1000 árboles se minimiza el error de prueba global.
- El error de entrenamiento con 800 árboles con `randomForest()` es el mismo que considerar 1000 árboles. La diferencia entre aumentar el número de árboles es la dismunición en el error de prueba.

Conclusiones

Primeramente, se muestra en la Tabla 13 una comparación entre los tres modelos. Se puede concluir que el modelo con menor error es DNN, seguido por RF y por último LogReg. Por tal motivo se selecciona al modelo DNN como el mejor para ajustar los datos de la tabla de validación.

	Training	Test error	CV
Red neurona profunda (DNN)	0.34	1.96	2.22
Random forest (RF)	3.36	3.14	3.65
Regresión logística regularizado (LogReg)	6.58	7.77	10.44

Tabla 13: Errores globales, en porcentaje, de los tres mejores modelos finales con distintos métodos. El error por validación cruzada (CV) se calculó para Red neuronal profunda con 4-*folds*, para random forest con 3-*folds*, y para Regresión logística regularizada con 10-*folds*. Para Random forest reportamos el error OOB global, en vez del error de entrenamiento.

La librería h2o presenta varias ventajas como la selección de modelos con una malla aleatoria bajo `h2o.grid` que disminuyen el tiempo de cómputo al no considerar todos los modelos. No obstante, tiene desventajas frente a otras librerías como fue en el caso de Random Forest. Esto se debió a que h2o ocupaba mucha memoria de la disponible, lo que provocó que varios modelos no terminaran de ajustarse, en especial los modelos con un incremento en el número de árboles.

Hay 54 observaciones, de las 11,000 de los datos de validación, para las cuales los tres modelos ajustan valores diferentes, 16 de estas observaciones se muestran en la Figura 7.

Por último, las tablas 14 y 15 corresponden a la matriz de confusión comparando los modelos LogReg y RF vs el modelo seleccionado como el mejor, DNN. Sobre estas tablas se pueden ofrecer los siguientes comentarios:

- En la diagonal de ambas tablas se observan valores grandes, estos valores representan observaciones de los datos de validación que están recibiendo la misma clasificación en cambos modelos.
- En la tabla 14 se pueden observar valores más grandes fuera de la diagonal en comparación con la tabla 15, lo que implica que las estimaciones difieren más entre los modelos DNN y LogReg que DNN y RF.
- La mayor confusión de la tabla 15 es 20, que corresponde a 20 observaciones que el modelo DNN clasifica como 4 y el modelo RF como 9. Hay 18 observaciones que el modelo DNN clasifica como 7 y el modelo RF como 9. Hay 13 observaciones que el modelo DNN clasifica como 8 y el modelo RF como 5.
- La mayor confusión de la tabla 14 es 52, que corresponde a 52 observaciones que el modelo DNN clasifica como 7 y el modelo LogReg como 9. Hay 41 observaciones que el modelo DNN clasifica como 4 y el modelo LogReg como 9. Hay 33 observaciones que el modelo DNN clasifica como 8 y el modelo RF como 5.
- En general, la mayor confusión se tiene entre los dígitos 7 y 9, y 4 y 9. Sin embargo, también existe confusión entre los dígitos 3 y 5, y 5 y 8.

	LogReg									
DNN	0	1	2	3	4	5	6	7	8	9
0	1075	1	7	5	0	7	8	1	7	0
1	0	1198	8	2	1	6	0	2	11	0
2	7	13	985	15	8	2	14	13	23	1
3	3	5	19	990	2	25	4	4	16	7
4	1	4	4	2	982	0	7	3	8	41
5	10	8	9	27	10	877	17	3	25	6
6	8	3	6	0	10	9	1045	1	4	3
7	2	7	6	7	10	4	0	1130	0	52
8	5	25	18	19	7	33	3	4	970	11
9	5	5	2	7	25	4	0	26	3	982

Tabla 14: Estimaciones por clase, de los datos `MNISTvalidate` , del modelo DNN vs LogReg.

	RF									
DNN	0	1	2	3	4	5	6	7	8	9
0	1096	0	4	1	1	0	3	0	6	0
1	0	1213	6	2	0	0	0	0	4	3
2	7	4	1052	3	1	0	2	6	4	2
3	1	2	9	1041	0	6	2	4	6	4
4	0	1	0	0	1024	0	3	2	2	20
5	4	2	2	7	1	955	9	1	7	4
6	5	2	3	0	1	7	1070	0	1	0
7	2	3	5	1	5	0	0	1184	0	18
8	1	3	4	5	2	13	3	0	1055	9
9	2	0	2	6	7	2	0	4	6	1030

Tabla 15: Estimaciones por clase, de los datos `MNISTvalidate` , del modelo DNN vs RF.

Anexo

DNN

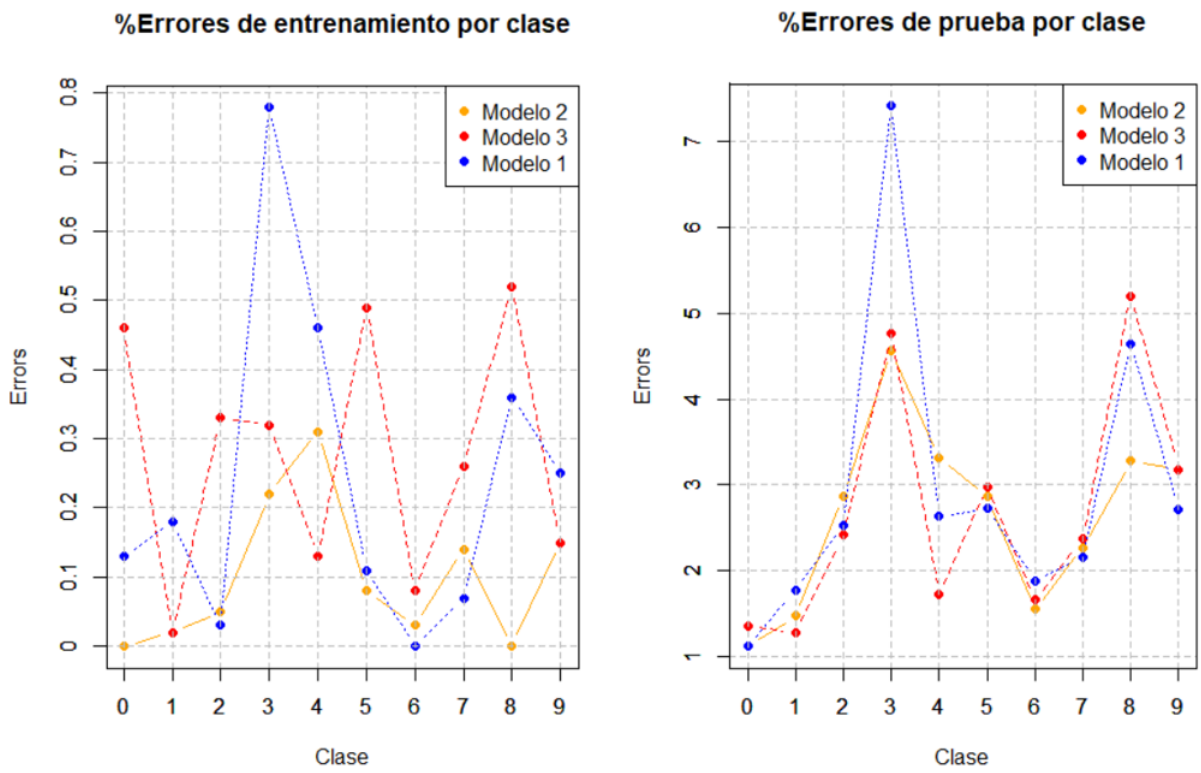

Figura 2: **Porcentaje** de errores por clase de los modelos simples *(DNN)*. En la izquierda se utilizó el conjunto de entrenamiento y a la derecha el conjunto de prueba.

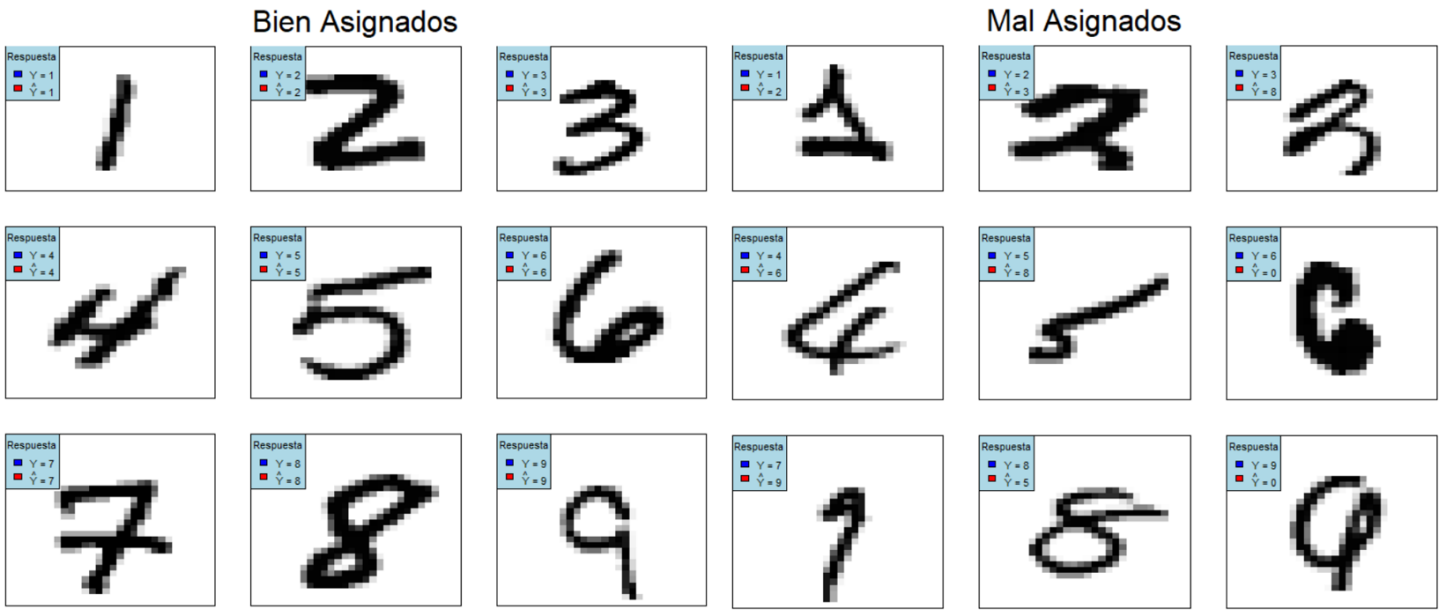

Figura 3: Matrices de números bien y mal asignados en el conjunto de **prueba** de acuerdo al modelo simple 2 *(DNN)*. En la matriz de 3×3 de la izquierda vemos los bien asignados y en la de la derecha los mal asignados.

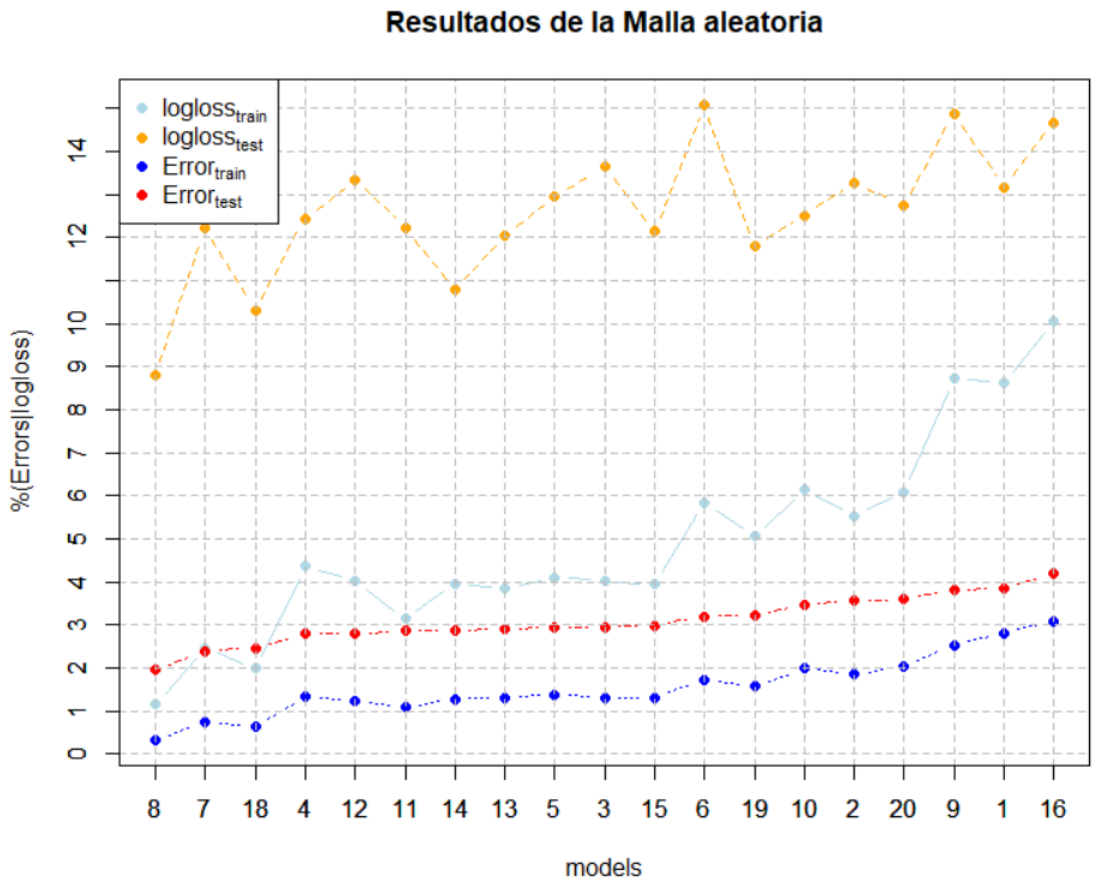

Figura 4: **Porcentaje** de errores y *logloss* para los conjuntos de entrenamiento y prueba de los modelos del 1 al 19 obtenidos de la malla mostrados en orden creciente. *Nota: El modelo 20 tenía un error muy grande y por motivos de escala no fue graficado.*

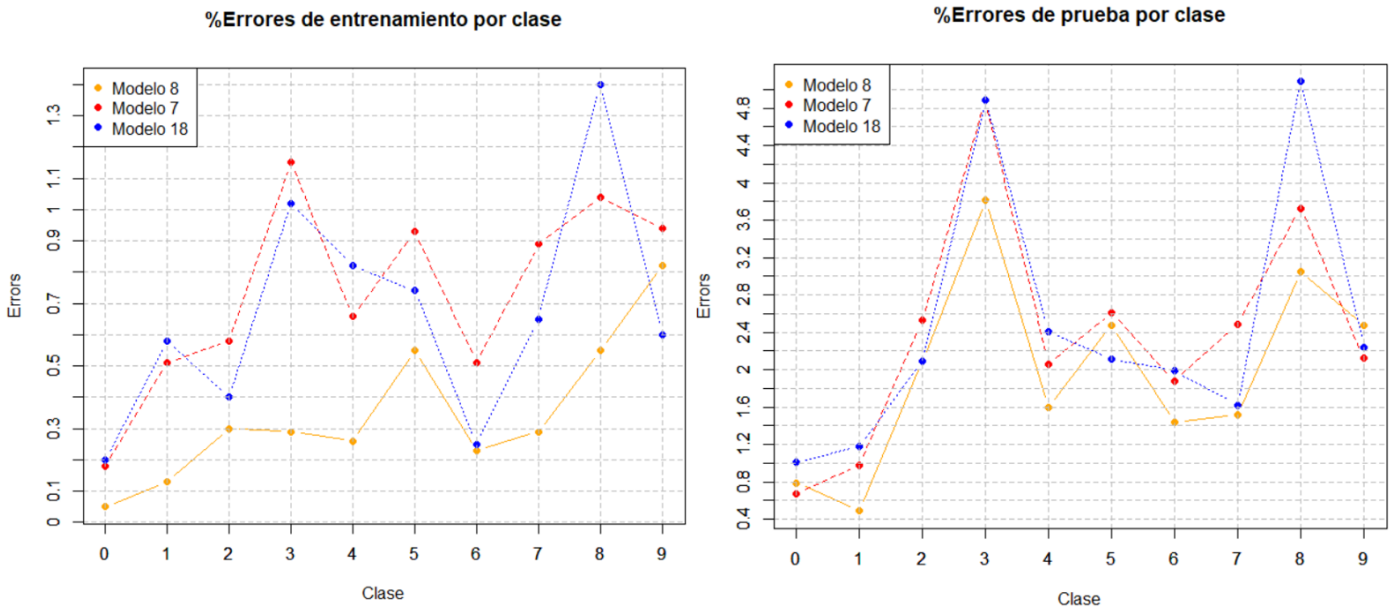

Figura 5: **Porcentaje** de errores por clase de los 3 mejores modelos de la malla *(DNN)*. En la izquierda se utilizó el conjunto de entrenamiento y a la derecha el conjunto de prueba.

Regresión logística

Figura 6: Matrices de 9 números bien ajustados y 9 números mal ajustados por el modelo multinomial con $\alpha = 0.2$. En la matriz de 3×3 de la izquierda se muestran predicciones que el modelo está haciendo de forma correcta y, en la derecha, predicciones incorrectas.

Random Forest

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`mtries=26`	1.22	1.51	3.45	4.93	2.94	4.61	2.01	3.76	5.01	4.95	3.41
`mtries=28`	1.35	1.51	3.25	4.98	2.79	4.36	1.83	4.00	4.91	4.97	3.37
`mtries=30`	1.27	1.56	3.55	4.98	2.84	4.36	1.96	3.90	4.91	5.07	4.42
`mtries=100`	1.43	1.65	5.41	6.81	3.63	7.63	5.42	6.12	5.97	6.34	4.99
`mtries=714`	2.60	2.03	6.12	7.00	4.86	6.70	3.67	7.25	6.91	7.23	5.40

Tabla 16: *Training errors*, en porcentaje, para distintos `mtries` con `h2o.randomforest()`, `ntrees=500` y `max_depth = 20` (default); excepto en el caso `mtries=100` se utilizó `max_depth = 10` para que pudiese terminar de compilar.

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`mtries=26`	0.67	1.77	3.08	7.00	3.21	3.85	3.21	3.57	4.75	3.77	3.48
`mtries=28`	0.67	1.96	3.41	6.79	2.98	3.73	3.32	3.57	4.30	3.53	3.42
`mtries=30`	0.67	1.86	3.19	7.00	3.44	4.10	3.54	3.57	4.52	3.77	3.56
`mtries=100`	0.45	1.67	5.05	8.27	3.89	6.83	7.31	6.82	4.75	5.30	5.00

Tabla 17: *Test errors*, en porcentaje, para distintos `mtries` con `h2o.randomforest()`, `ntrees=500` y `max_depth = 20` (default).

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`max_depth=21`	1.32	1.49	3.35	5.17	2.97	4.28	1.99	3.83	4.86	5.04	3.41
`max_depth=20`	1.30	1.56	3.40	4.93	2.84	4.53	1.91	4.02	5.01	4.95	3.42
`max_depth=10`	1.38	1.51	5.18	6.68	3.89	7.22	2.77	5.28	6.42	6.71	4.66

Tabla 18: *Training errors*, en porcentaje, para distintos `max_depth` con `h2o.randomforest()`, `mtries=28` y `ntrees=500`.

Parámetro*Clase*	0	1	2	3	4	5	6	7	8	9	Error Global
`max_depth=21`	0.67	1.77	3.41	6.79	2.98	3.85	3.43	3.46	4.75	3.89	3.44
`max_depth=20`	0.67	1.77	3.41	6.79	2.98	3.85	3.43	3.46	4.75	3.89	3.49
`max_depth=10`	0.90	1.86	4.95	8.70	4.58	6.09	4.21	5.41	5.77	4.83	4.70

Tabla 19: *Test errors*, en porcentaje, para distintos `max_depth` con `h2o.randomforest()`, `mtries=28` y `ntrees=500`.

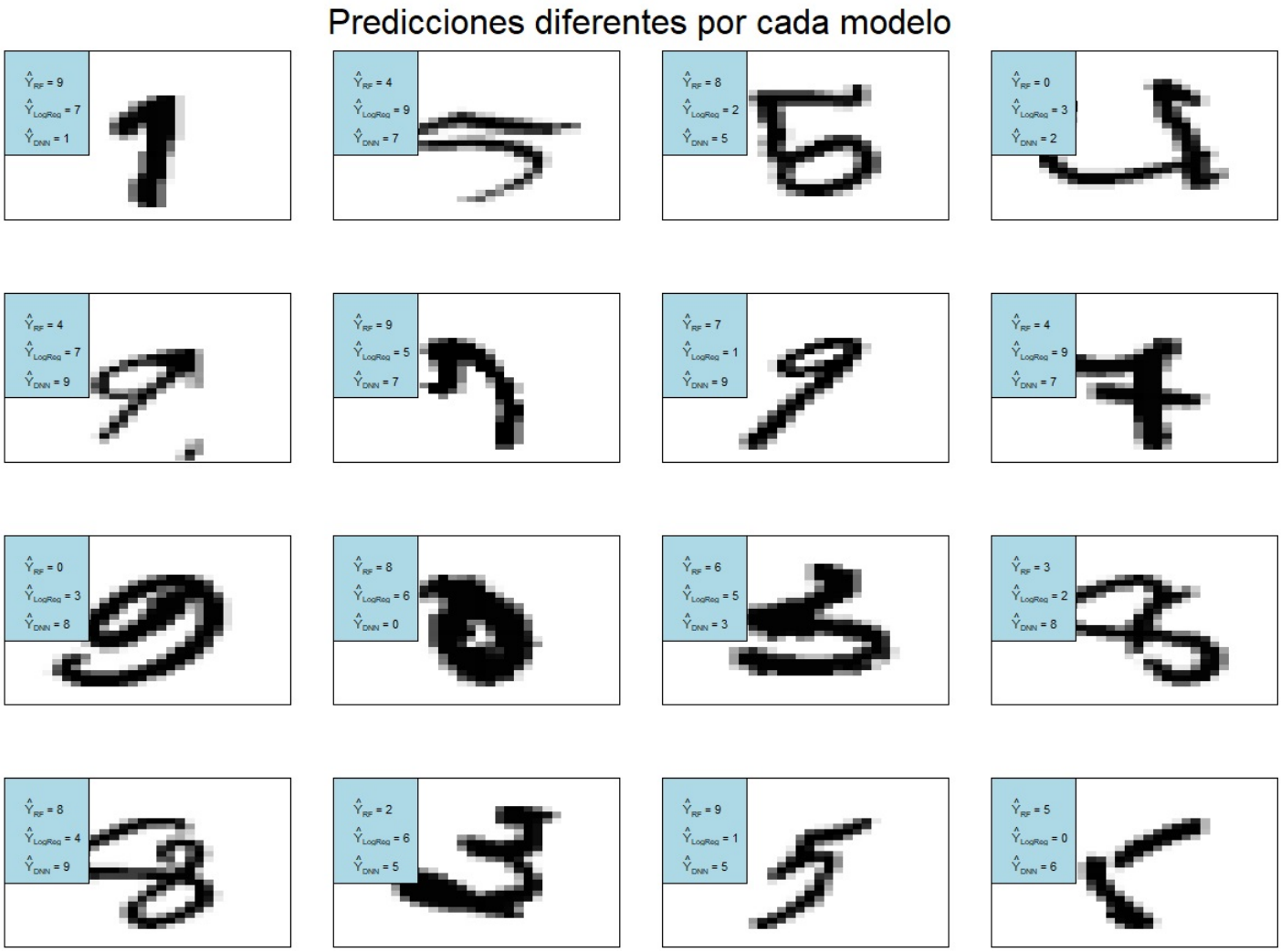

Figura 7: 16 observaciones de 54 de los datos de validación para las cuales los tres modelos seleccionados predicen diferente.