Proyecto Final

González Feria Juan Rosendo Gonźalez González Elvira
Miranda Peñafiel Melissa Sofía Nicolas Mata Jesús
Pacheco Martínez Mariana

10 de junio de 2020

Ejercicio 1

La base de datos correspondiente a MNIST la hemos particionado de forma aleatoria en tres conjuntos: *train*, *test*, y *validate* de *40000*, *9000* y *11000* registros respectivamente. Los dos primeros, `MNISTtrain.csv` y `MNISTtest.csv` están disponibles en *classroom*. El tercero `MNISTvalidate.csv` solo contiene a las variables predictoras ($x's$) y no a la variable clase (y), la hemos puesto a disposición también en *classroom* para que cada equipo construya dos vectores:

a) Uno con los valores predichos utilizando el modelo de *DNN* final reportado y entrenado con `MNISTtrain.csv`,

b) Otro con los valores predichos por un modelo de regresión logística no regularizado ajustado también con `MNISTtrain.csv`.

Utilice las bases de datos `MNISTtrain.csv` y `MNISTtest.csv` y realice lo siguiente:

1. Ajuste o entrene una red neuronal profunda, *DNN*, para clasificar al conjunto de observaciones con un error de prueba lo más bajo posible.

2. Ajuste o aprenda un modelo de regresión logística no regularizado.

Modelo DNN

Tras elaborar un total de 15 mallas aleatorias obtuvimos alrededor de 240 modelos para empezar a seleccionar el que mejor clasificara los datos según algunos criterios que mencionaremos más adelante.

El modelo final ajustado contó con 20 épocas, 2 capas de 253 y 69 nodos respectivamente, también existe un parámetro de regularización $l_1 = 0.00043$.

Se presentaron los siguientes errores de predicción:

Error	Train	Test	Cross-validation
Global	1.58	2.79	3.13
Grupo 0	0.64	0.79	1.81
Grupo 1	0.67	0.79	1.45
Grupo 2	1.61	3.19	3.17
Grupo 3	1.68	5.30	4.46
Grupo 4	1.05	1.37	2.94
Grupo 5	2.22	3.60	4.36
Grupo 6	0.89	2.33	1.93
Grupo 7	2.63	3.57	3.30
Grupo 8	1.90	3.28	4.08
Grupo 9	2.58	3.89	4.03

Cuadro 1: Tasas de error presentadas en porcentajes para el modelo ajustado de DNN. Cross-validation se realizó usando *n folds* = 5.

Mientras que las estadísticas del logloss fueron las siguientes

	Train	Test	Cross-validation
Logloss	0.0544	0.0929	0.1061

Cuadro 2: Medidas de logloss para el modelo ajustado de DNN. Cross-validation se realizó usando *n folds* = 5.

Como mencionamos anteriormente, elaboramos varias mallas aleatorias, cada vez cambiando los parámetros según lo que se observaba de los modelos que parecían tener el mejor desempeño. Esto último se realizó, principalmente, tomando en cuenta el error de clasificación para el conjunto *train* y *test*, así como el valor de logloss para ambos. Se elaboraron gráficas de diagnóstico para cada malla y se observó si existía un sobreajuste tomando como referencia las métricas ya mencionadas.

A continuación presentamos una de las mallas; esta consta de 20 modelos diferentes, y nos permitió analizar varios aspectos:

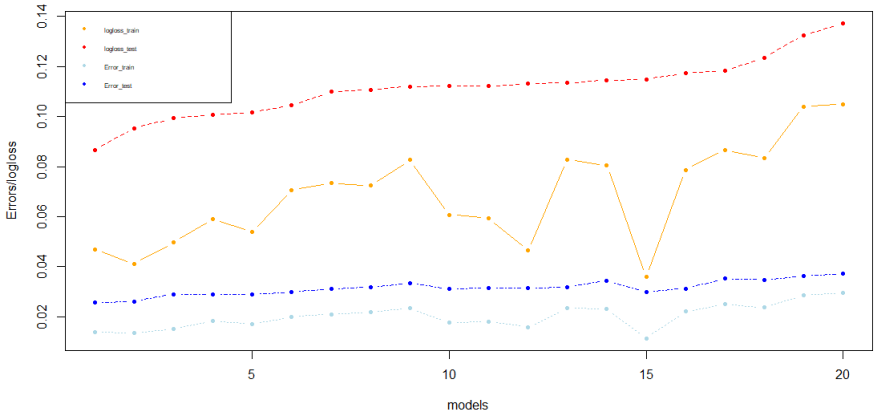

Figura 1: Gráfica diagnóstico de una malla de 20 modelos.

Model	Run Time Mins	layers	epochs	l_1	logloss train	loglss test	Error train	Error test
1	2.97	201-186	50	0.00025	0.0469	0.0867	0.014	0.0258
2	4.06	169-184-162	50	2e-04	0.0411	0.0953	0.0136	0.026
3	3.06	178-130	50	3e-04	0.0497	0.0993	0.0152	0.0291
4	3.16	207-145-188	50	0.00042	0.059	0.1007	0.0184	0.0288
5	2.36	141-138-153-122	50	0.00032	0.054	0.1016	0.0171	0.0291
6	4.29	211-211-203	50	0.00071	0.0707	0.1044	0.02	0.0299
7	3.28	192-133-121-176	50	0.00063	0.0734	0.1098	0.0211	0.0312
8	2.9	141-138-153-122	50	0.00075	0.0724	0.1106	0.0218	0.0319
9	2.42	201-186	50	0.00067	0.0826	0.112	0.0234	0.0334
10	2.24	153	50	0.00031	0.0608	0.1121	0.0176	0.0311
11	1.59	123-194-145	50	0.00029	0.0595	0.1121	0.0182	0.0316
12	2.07	185-169	50	0.00014	0.0464	0.113	0.0159	0.0316
13	2.69	141-138-153-122	50	0.00087	0.0827	0.1134	0.0236	0.0319
14	2.67	173-137-170	50	0.00088	0.0805	0.1145	0.0232	0.0344
15	1.84	148-200-122	50	0.00012	0.036	0.1148	0.0114	0.0299
16	2.21	152	50	5e-04	0.0787	0.1173	0.0221	0.0313
17	2.62	185-190-124	50	0.00083	0.0866	0.1183	0.0251	0.0352
18	3.41	204	50	0.00062	0.0835	0.1235	0.0237	0.0348
19	2.54	219	50	0.00081	0.1039	0.1325	0.0287	0.0364
20	2.22	165	50	0.00095	0.1049	0.1371	0.0296	0.0372

Cuadro 3: Especificaciones de los 20 modelos obtenidos de un *h2o.grid* para DNN, los errores están en decimales, *stopping_metric* es *logloss* para clasificación.

Por ejemplo, para esta malla vimos que, aunque el error del primer modelo es el más bajo, la distancia entre su *logloss_trn* y *logloss_tst* es muy grande y esto puede indicar un sobre ajuste del modelo, y viendo solo la gráfica diríamos que los mejores modelos son el 9 y 13; esto se debe a que tienen las menores distancias entre *logloss_trn* y *logloss_tst*, conservando un error bajo, a diferencia del 19 o 20 donde conserva una distancia pequeña, pero crece el error. Decidimos no descartar el modelo 1 ya que era el que conservaba el error más bajo y pensamos que esto

podía deberse a que eran muchas épocas.

Después de este análisis de la gráfica, fuimos a la tabla para ver similitudes entre los mejores modelos, en este caso el 1, 9 y 13. Comparando el 1 y el 9, notamos que tenían exactamente el mismo número de capas y nodos, épocas, el tiempo que tardaban era similar, y en lo único que diferían, además del error, era el parámetro *l_1*, por lo que empezamos a hacer intentos variando *l_1* en ese intervalo (0.00025,0.00067). Finalmente, al ver que el 13 contaba con más capas y nodos y que su *l_1* no estaba en ese intervalo, lo descartamos, hicimos este proceso de comparación de gráficas y tablas en cada uno de los 15 intentos, para obtener las especificaciones de los mejores modelos, además de tomar en cuenta otros parámetros que mencionamos a continuación.

Entre los diferentes parámetros que modificamos se encuentran:

- Cantidad de capas y de nodos, respectivamente.
- *epochs*: Total de iteraciones sobre todo el conjunto *train*. Las épocas las modificamos varias veces, porque sabemos que pocas pueden darnos un mal modelo y muchas pueden sobre ajustarlo, es por esto que las variamos entre 20-100 en cada malla. Y finalmente el mejor modelo ocupó 20.
- *l1*: Parámetro de regularización. Hicimos un intento aumentando este parámetro y al ver que el error de logloss se disparaba muchísimo (pasaba de 8 % a 29 %), decidimos que la mejor opción era mantenerlo bajo.
- *stopping_tolerance*: Tolerancia relativa al error, detiene el programa si el error relativo no mejora al menos *k*. Este parámetro se mantuvo en 0.05 para todos los modelos.
- *stopping_metric* (StopM): Métrica de referencia para detener el programa a partir del *stopping_tolerance*. Para esta variable vimos que había dos métodos para clasificación: *misclassification* y *logloss* para clasificación, y notamos que al usar *logloss* podíamos conservar un error bajo y hacer que la diferencia entre *logloss_trn* y *logloss_tst* disminuyera. Para la malla anterior utilizamos *logloss*.
- *score_interval* (ScoreI): Tiempo que tarda en generar cada iteración. Este parámetro solo lo variamos dos veces, una vez con 3 y otra con 5, y al darnos cuenta que no había un cambio significativo en el error, finalmente lo dejamos en 2. Para la malla anterior este parámetro es 3.
- *activation* (Act): Función de activación. Al llevar varios intentos y ver que no mejoraba mucho el error, decidimos indagar un poco en otros parámetros y corrimos una malla variando esta función con *Rectifier, Rectifier with dropout, Tanh y Tanh with dropout*, pero vimos que la función por default (*Rectifier*), seguía siendo la que arrojaba los mejores modelos y la ocupamos para el modelo final y los siguientes.

Durante el proceso de ajuste notamos que los mejores modelos contaban con 2-3 capas, y que las primeras contaban con más nodos que las últimas, para el *l1* siempre se mantuvo entre 0.0002 y 0.0008; en las épocas fue un poco difícil decidir cuántas eran las necesarias, ya que los mejores modelos variaban entre 20, 50 y 100 épocas, de igual forma esto sucedió con el parámetro *stopping_metric*, porque a veces el menor error se obtenía con *misclassification* y otras veces con *logloss* (solo variando este parámetro).

Estas son las especificaciones completas del mejor modelo obtenido

Modelo	Capas	Épocas	l1	StopM	ScoreI	Act	nfolds	Run Time Mins
Modelo_DNN	253-69	20	0.000043	*logloss*	2	*Rectifier*	5	18.64

Cuadro 4: Especificaciones de los parámetros para el modelo final de DNN.

Modelo de regresión logística sin regularizar

Mientras que el modelo DNN tiene bastantes parámetros para modificar, el caso de la regresión logística sin regularizar no es así, siendo `max_iterations` el único parámetro a variar. `max_iterations` es el número máximo permitido a la base train para ser iterada.

Teniendo eso en mente, se corrieron un total de 5 modelos[1] variando dicho parámetro. A continuación se muestran los resultados obtenidos para estos 5 modelos:

Modelo	`max_it`	Train [%]	Test [%]	CV 5 folds [%]	Logloss Train	Devianza Residual Train	Logloss Test	Devianza Residual Test
1	50	5.47	8.56	9.04	0.1971	15769.32	0.3697	6741.77
2	100	5.04	9.27	9.83	0.1797	14383.78	0.4349	8035.30
3	1000	4.66	10.10	10.92	0.1688	13510.76	0.5680	12188.01
4	10000	4.65	10.07	11.01	0.1686	13494.19	0.5796	13034.85
5	100000	4.65	10.07	11.01	0.1686	13494.19	0.5796	13034.85

Cuadro 5: Características de los 5 modelos que se hicieron con regresión logística. El único parámetro que se varió fue `max_iterations`. Se muestran los errores globales de train, test y CV con 5 folds. El tiempo de ejecución de los cinco modelos es de *68 minutos*.

Se puede observar en el cuadro anterior que los valores de logloss tienden a ciertas cantidades. Por un lado, para el conjunto train el valor del logloss baja, mientras que para el conjunto test el logloss aumenta. Se tiene la siguiente gráfica de comparación de los valores de logloss para los conjuntos train y test para los 5 modelos implementados:

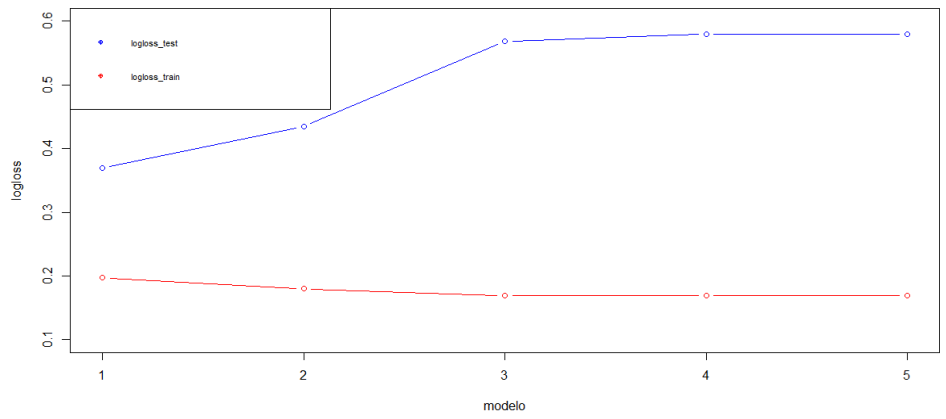

Figura 2: Gráfica de logloss de los conjuntos train y test en función del modelo utilizado. Cada modelo difiere en el número de iteraciones máximo.

[1] En realidad se corrieron 7 modelos, pero no se alcanzaron a guardar por problemas con la computadora.

Podemos observar que el primer modelo es el que tiene el logloss más bajo, pues conforme aumenta el número máximo de iteraciones, el logloss del conjunto test aumenta. Otra cosa que tiene el primer modelo es que la diferencia de los valores de logloss para train y test es la más pequeña.

Fueron estos detalles los que nos hicieron quedarnos con el modelo 1. Para este modelo seleccionado se obtuvieron las siguientes tasas de error de clasificación para cada uno de los grupos:

Error	Train [%]	Test [%]	Cross Validation [%]
Global	5.47	8.56	9.04
Grupo 0	2.04	2.7	4.35
Grupo 1	2	3.53	2.85
Grupo 2	6.92	9.89	11.96
Grupo 3	8.1	13.47	12.37
Grupo 4	4.7	6.53	8.11
Grupo 5	8.51	13.66	13.97
Grupo 6	2.75	5.98	5.29
Grupo 7	4.41	6.93	7.54
Grupo 8	8.86	14.71	14.57
Grupo 9	7.16	9.31	10.61

Cuadro 6: Tasas de error clasificación para cada uno de los grupos. El tiempo de ejecución del modelo fue de *45 segundos*.

Respecto al coste computacional, la aplicación de los modelos de Regresión Logística requirieron menor poder de cómputo en comparación a los modelos para la Red Neuronal Profunda.

Comparación Modelo Red Neuronal VS Regresión Logística sin Regularizar

Se muestran en el siguiente cuadro, las características de los mejores modelos implementados:

Modelo	Cápas	Épocas	l1	StopM	ScoreI	Act	nfolds	RunTime Min
Modelo_DNN	253-69	20	0.000043	*logloss*	2	*Rectifier*	5	18.64

Cuadro 7: Especificaciones de los parámetros para el modelo final de DNN.

Modelo	`max_it`	Train [%]	Test [%]	CV 5 folds [%]	Logloss Train	Devianza Residual Train	Logloss Test	Devianza Residual Test
1	50	5.47	8.56	9.04	0.1971	15769.32	0.3697	6741.77

Cuadro 8: Características del modelo final con la aplicación de Regresión Logística sin Regularizar.

A continuación se muestran las tasas de error para cada uno de los grupos:

	Red Neuronal			Regresión Logística		
Error	Train	Test	Cross Validation	Train	Test	Cross Validation
Global	1.58	2.79	3.13	5.47	8.56	9.04
Grupo 0	0.64	0.79	1.81	2.04	2.70	4.35
Grupo 1	0.67	0.79	1.45	2.00	3.53	2.85
Grupo 2	1.61	3.19	3.17	6.92	9.89	11.96
Grupo 3	1.68	5.30	4.46	8.10	13.47	12.37
Grupo 4	1.05	1.37	2.94	4.70	6.53	8.11
Grupo 5	2.22	3.60	4.36	8.51	13.66	13.97
Grupo 6	0.89	2.33	1.93	2.75	5.98	5.29
Grupo 7	2.63	3.57	3.30	4.41	6.93	7.54
Grupo 8	1.90	3.28	4.08	8.86	14.71	14.57
Grupo 9	2.58	3.89	4.03	7.16	9.31	10.61

Cuadro 9: Tasas de Error por Grupo para los dos mejores modelos implementados. En ambos casos el numero de folds para validación cruzada es de 5.

Podemos observar que el mejor modelo de la red neuronal presenta las tasas de error más bajas con respecto al mejor modelo de la regresión logística sin regularizar. Se tienen ahora los siguientes valores para el logloss:

	Red Neuronal			Regresión Logística		
	Train	Test	Cross Validation	Train	Test	Cross Validation
Logloss	0.0544	0.0929	0.1061	0.1971	0.3397	0.3778

Cuadro 10: Comparación Logloss para los mejores modelos. En ambos casos el número de folds es 5.

Nuevamente podemos ver que el valor del logloss es mucho menor en la red neuronal implementada que en el modelo de regresión logística.

Ejercicio 2

Utilice la base de datos `MNISTvalidate.csv` de 11,000 registros y calcule los dos vectores siguientes.

i) Un vector de 11,000 entradas con los valores predichos obtenidos con el modelo final *DNN* reportado. Guárdelo en un archivo con nombre que registre el número de equipo y el apellido del primer integrante del equipo.

ii) Un vector de 11,000 entradas con los valores predichos por el modelo de regresión logística ajustado y reportado.

iii) Julio calculará el error de predicción validado, global y por grupo, de los modelos al comparar su vector de valores predichos vs el vector de valores observados retenidos.

Modelo DNN

Resultado anexo en **Equipo4_GnzlzFeria_DNN_pred**

Modelo de regresión logística

Resultado anexo en **Equipo4_GnzlzFeria_RegLog_pred**

Apéndice

DNN

Como para el DNN teníamos muchísimas mallas para escoger, como se mencionó anteriormente, se seleccionaron los mejores modelos de cada malla y se realizó una comparación entre estos

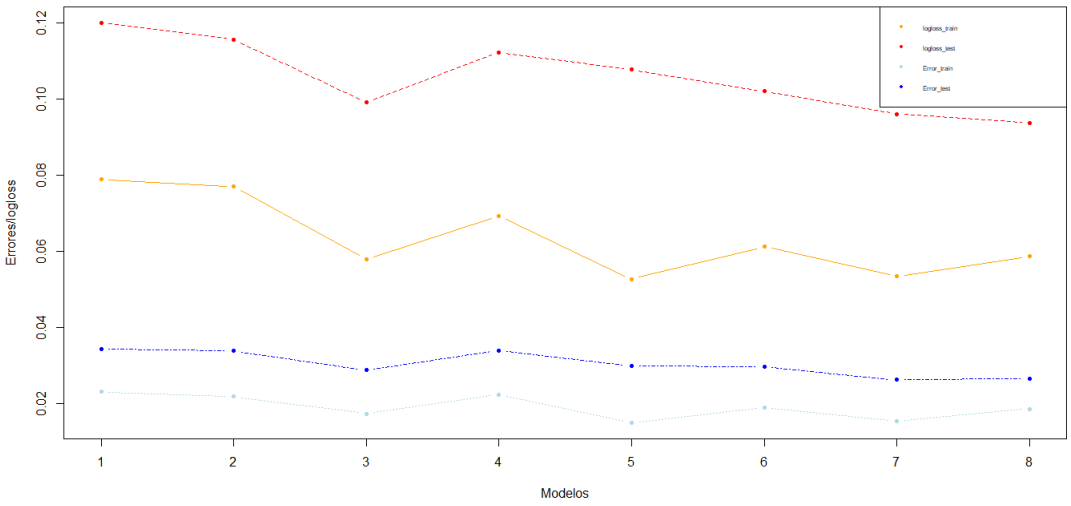

Figura 3: Gráfica diagnóstico de los mejores 8 modelos obtenidos.

Modelo	Run time	Capas	Épocas	l1	logloss train	logloss test	Error train	Error test	stopping metric
1	3.41	201-186	50	0.00067	0.0789	0.1200	2.31	3.44	misclass
2	3.53	201-186	50	0.00067	0.0770	0.1157	2.18	3.38	logloss
3	1.68	145-123-157	100	0.00034	0.0579	0.0991	1.73	2.88	misclass
4	1.41	145-123-157	100	0.00034	0.0692	0.1123	2.23	3.39	logloss
5	5.80	190-174-190	100	0.00043	0.0527	0.1078	1.50	2.99	misclass
6	2.62	190-174-190	100	0.00043	0.0612	0.1021	1.89	2.97	logloss
7	4.27	253-69	20	0.00043	0.0534	0.0961	1.53	2.62	misclass
8	3.29	253-69	20	0.00043	0.0587	0.0937	1.86	2.66	logloss

Cuadro 11: Especificaciones de los mejores 8 modelos obtenidos de DNN. *Run time* está dado en minutos, los errores están presentados en porcentajes y para *stopping metric* misclass es *"misclassification"*.

Finalmente, nuestro criterio para escoger el modelo que haría las predicciones fue tomar los que tuvieran los logloss más bajos y que la diferencia entre el logloss de *train* contra el de *test* fuese mínima.

Decidimos que la medida de logloss fuera la decisiva ya que ésta penaliza al modelo por las clasificaciones incorrectas y las tasas de error que se obtuvieron, por lo general, se movían entre los $0.02 - 0.04$.

Modelo	Diferencia
Modelo 1	0.0411
Modelo 2	0.0387
Modelo 3	0.0412
Modelo 4	0.0431
Modelo 5	0.0551
Modelo 6	0.0409
Modelo 7	0.0427
Modelo 8	0.035

Cuadro 12: Diferencias entre logloss de *test* y de *train* de los 8 mejores modelos.

Como se puede ver en la tabla anterior el modelo que presentaba la menor diferencia fue el modelo 8, entonces este fue el que se usó para ajustar y predecir los datos.

Como comentario adicional para el DNN, podemos agregar que la mayoría de las mallas eran de 20 modelos y estas tardaban en correr en promedio de 40 a 60 min. Una vez seleccionados los mejores modelos, sin *nfolds*, estos tardaban de 2 a 4 minutos en correr (individualmente), y aplicando *nfolds*=5, el tiempo rondaba entre los 11 y los 18 minutos por modelo. Nos parece importante mencionar esto ya que estamos acostumbrados a hacer programas que tardan, literalmente, segundos en correr y haciendo este proyecto pudimos darnos cuenta de la complejidad de las redes neuronales, y que la teoría detrás de estas es tan "pesada"que hasta a una computadora le cuesta trabajo procesarla.

A continuación presentamos dos ejemplos de algunas de las mallas que corrimos:

```
(hidden_opt = lapply(1:100, function(x)120+sample(100,sample(4), replace=TRUE)))
(l1_opt = seq(1e-4,1e-3,1e-5))
(hyper_params <- list(hidden = hidden_opt, l1 = l1_opt))
search_criteria = list(strategy = "RandomDiscrete",
                       max_models = 20,
                       seed=915)


system.time(
  model_gridz <- h2o.grid("deeplearning",
                          grid_id = "mygrid",
                          hyper_params = hyper_params,
                          search_criteria = search_criteria,
                          x = x,
                          y = y,
                          distribution = "multinomial",
                          training_frame = train,
                          validation_frame = test,
                          score_interval = 3,
                          epochs = 50,
                          stopping_rounds = 3,
                          stopping_tolerance = 0.05,
                          stopping_metric = "logloss"))
```

Figura 4: Código con especificaciones de la malla para un *h2o.grid*

Para esta malla cambiamos a un mínimo de 120 nodos por capa, reducimos el intervalo para *l1*, aumentamos *score_interval* para ver si al aumentar el tiempo mejoraba el error, bajamos las épocas a 50 y modificamos *stopping_metric* a "logloss".

```
(hidden_opt = lapply(1:100, function(x)10+sample(300,sample(5), replace=TRUE)))
(l1_opt = seq(1e-5,1e-3,1e-5))
(epoch_opt=20)
(activation_opt = c("Rectifier", "Rectifier with dropout", "Tanh",
                    "Tanh with dropout"))
hyper_params <- list(hidden = hidden_opt, l1 = l1_opt, epochs=epoch_opt,
                     activation= activation_opt)
search_criteria = list(strategy = "RandomDiscrete",
                       max_models = 20,
                       seed=1)

system.time(
      model_gridz<- h2o.grid("deeplearning",
                          grid_id = "randomg1",
                          hyper_params = hyper_params,
                          search_criteria = search_criteria,
                          x = x,
                          y = y,
                          distribution = "multinomial",
                          training_frame = train,
       #                  nfolds = 5,
                          seed = 1,
                          score_interval = 2,
                          stopping_rounds = 3,
                          stopping_tolerance = 0.05,
                          stopping_metric = "misclassification"))
```

Figura 5: Código con especificaciones de la malla para un *h2o.grid*

Para esta malla decidimos aumentar el máximo de nodos y disminuir el mínimo de los mismos, también aumentamos las posibles capas a 5, además de bajar las épocas a 20 y variar la función de activación para ver si esta producía algún cambio significativo en el desempeño de los modelos.

Regresión Logística sin regularizar

Se corrieron originalmente 7 modelos en los que el único parámetro que se varió fue el de `max_iterations`, aunque hay que decir que también se intentaron variar otros parámetros más como por ejemplo el parámetro `solver, beta_epsilon, etc`. Sin embargo el variar estos últimos no presentaba mejoras al modelo y hacian que el tiempo de computo fuera mayor. A continuación se tiene el código que se implemento:

```
system.time(modelos <- lapply(c(50, 100, 1000, 1e4, 1e5),
                              function(iters){
                                  h2o.glm(x=x,y=y,
                                          training_frame = train,
                                          nfolds = 5,
                                          family = "multinomial",
                                          lambda = 0,
                                          seed = 1,
                                          max_iterations = iters)}))
```

Figura 6: Código Implementado para la Regresión Logística sin regularizar