Ejercicios Neural Network

Autores: Daniel Ferreira Zanchetta y Lais Silva Almeida Zanchetta

Enunciado parte 1) Predicción sobre los datos Musk utilizando:

- Multinom con nnet
- MultiLayer Perceptron con Keras

Para realizar estos ejercicio, hemos decido probar un número de modelos que creíamos ser significativo para realizar cierta comparación. Entendemos y asumimos que, sobretodo, habría una infinidad más de modelos que podríamos construir y probar. Sin embargo para algunos modelos hemos percibido no tener mucha capacidad de máquina, y otras veces también nos encontramos con errores que no sabíamos solventar. Un ejemplo del error es lo que remarco en rojo en la parte 2 de este informe (pero dejo esto aparte de momento para focalizar en la parte 1).

En relación a los datos de MUSK, tratamos de montar dos tipos de sets de pre-procesamiento y, por encima de estos dos distintos sets, probamos 12 modelos distintos. Los set de pre-procesamiento son los que llamamos de #1 y #2, explicados a continuación:

Pre Procesamiento #1) Eliminación de las dos primeras variables; eliminación de duplicados; normalización de variables numéricas y de la variable Class; división de 70% de muestreo para training y 30% para tests;

Pre Procesamiento #2) Mismas características del uno pero aplicando el PCA. Al aplicar el PCA hemos llegado al número de 25 componentes principales, con los que hemos usado para los modelos.

En la tabla que presentamos a continuación, hemos dividido los resultados basado en los outputs que hemos obtenido en las ejecuciones. Es decir, del ID 1 al ID 6 tenemos el Error rate de cada modelo. Por otro lado, del ID 7 al ID 12, tenemos el accuracy rate obtenido con cada modelo, que es lo que nos ha devuelto las funciones de Keras.

Después de ejecutar los distintos modelos, lo que entendemos que tiene el mejor resultado de predicción es el marcado en el ID 8 para el set de datos del Pre Procesamiento #1 – MLP con Keras y 3 capas, utilizando como función de activación ReLU. Si que es verdad que hemos notado que el accuracy de training (99.13%) es más pequeño si comparado con otros modelos MLP. No obstante, este ha sido el modelo que tuvo accuracy en Test más grande, así que lo hemos elegido basado en este criterio.

Con la intención de probar otros modelos que no sean de Neural Networks, también probamos lineares como LDA, QDA, y también un modelo utilizando Random Forest. Nos ha parecido interesante el resultado del ID 5, para la ejecución sin CV LOO, pero aun así el ID 8 es mejor para este ejercicio.

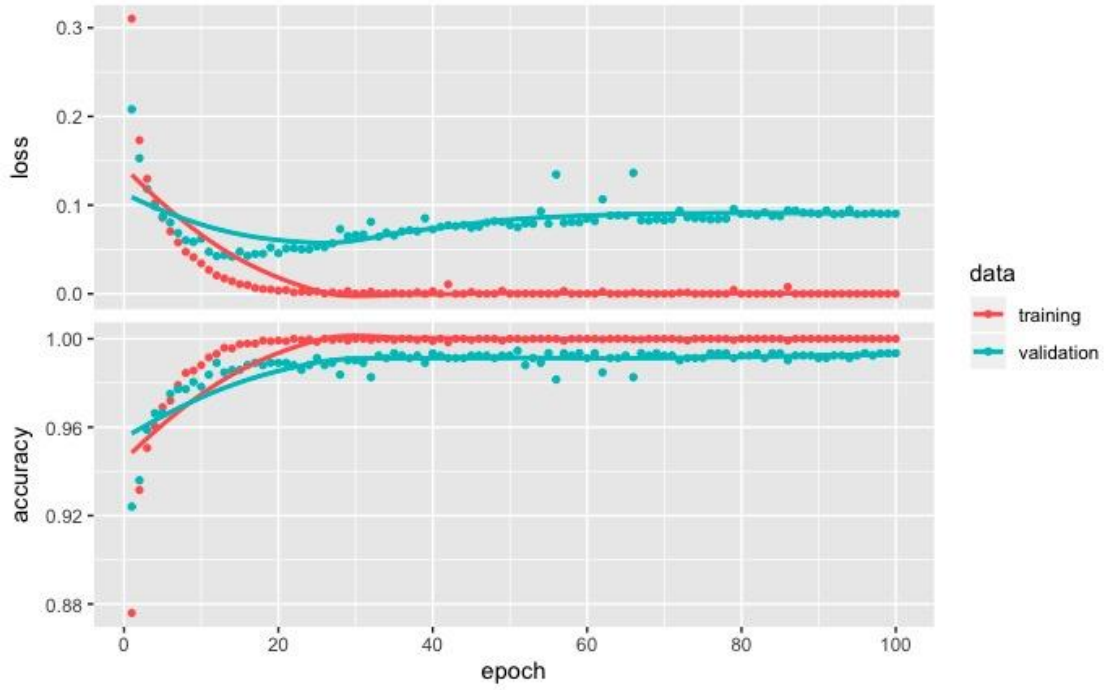
ID	Resumen del Modelo	Configuración	Medido en Error rate (Resultados del modelo para cada característica de pre-procesamiento)	
			Set de Datos	
			Pre Procesamiento #1	Pre Procesamiento #2
1	Multinom con nnet	decay = 0 maxit = 100	Training: 15.95% Test: 5.06%	Training: 15.95% Test: 8.50%
		decay = 0.1 maxit = 300	Training: 15.95% Test: 4.65%	Training: 15.95% Test: 8.55%
		decay = 0.001 maxit = 500	Training: 15.95% Test: 5.01%	Training: 15.95% Test: 8.50%
2	NNET con 1 capa oculta	decay = 0 maxit = 100	Training: 10.39% Test: 11.69%	Training: 8.87% Test: 8.75%
3	NNET con 10 capas ocultas	decay = 1 maxit = 100	Training: 0.32% Test: 1.36%	Training: 2.30% Test: 2.93%
4	LDA	Sin Cross Validation	Training: 5.23% Test: 5.87%	Training: 8.94% Test: 9.06%
		Con Cross Validation LOO	Training: 5.88% Test: 6.02%	Training: 8.98% Test: 8.86%
5	QDA	Sin Cross Validation	Training: 1.02% Test: 2.93%	Training: 9.68% Test: 10.43%
		Con Cross Validation LOO	Training: 3,53% Test: 6.18%	Training: 10.09% Test: 9.36%
6	Random Forest	ntree = 200	4,35%	4.30%
			Medido en Accuracy	
7	MLP con Keras y 1 layer	Función activación: ReLU Output: Softmax Epochs=100	Training: 99.46% Test: 99.19%	Training: 97.83% Test: 98.18%
8	MLP con Keras y 3 layers	Función activación: ReLU Output: Softmax Epochs=100	Training: 99.13% Test: 99.29%	Training: 98.05% Test: 97.42%
9	MLP con Keras y 1 layer	Función activación: ReLU Output: Sigmoid Epochs=100	Training: 99.35% Test: 99.22%	Training: 97.67% Test: 98.30%
10	MLP con Keras y 3 layers	Función activación: ReLU Output: Sigmoid Epochs=100	Training: 99.40% Test: 99.24%	Training: 97.61% Test: 97.11%
11	MLP con Keras y 5 layers	Función activación: ReLU Output: Sigmoid Epochs=100	Training: 99.02% Test: 98.66%	Training: 97.40% Test: 97.47%

ID	Resumen del Modelo	Configuración	Medido en Error rate (Resultados del modelo para cada característica de pre-procesamiento)	
			Set de Datos	
			Pre Procesamiento #1	Pre Procesamiento #2
12	MLP con Keras y 4 layers	Función activación: ReLU Output: Sigmoid Epochs=100	Training: 99.02% Test: 98.89%	Training: 97.45% Test: 97.70%

La construcción del código para el modelo del ID 8 ha sido lo siguiente:

```
library(dplyr)
library(keras)

#### Pre-processing antes de ejecutar Keras ####
#Train data and test data sin la variable de respuesta class
train_musk_x <- train_musk[, -167]
test_musk_x <- test_musk[, -167]

levels.class <- length(levels(train_musk$class))
train_musk_y_num <- as.integer(train_musk$class)
test_musk_y_num <- as.integer(test_musk$class)
train_musk_y <- to_categorical(train_musk_y_num-1, levels.class)
test_musk_y <- to_categorical(test_musk_y_num-1, levels.class)

#Predictors transformados a matrix, pues Keras no funciona con data.frame
matrix_train_musk <- as.matrix(train_musk_x)
matrix_test_musk <- as.matrix(test_musk_x)

#### Keras con 3 layers y 1 neurona con Softmax y utilizando como función
 de activación el ReLU ####

per2 <- keras_model_sequential()
per2 %>% layer_dense(units = 64, activation = "relu",
              input_shape = c(ncol(train_musk_x))) %>%
        layer_dense(units = 32, activation = "relu") %>%
        layer_dense(units = 16, activation = "relu") %>%
        layer_dense(units = levels.class, activation = "softmax") #2 neur
onas de output

# Ahora compilamos con validation metric (loss function = binary_crossent
ropy) y un optimizador (accuracy)
per2 %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy") #evaluación de lo buena que es la red neuronal
)
```

```
per2

# Entrenando el modelo de red neuronal con 1 layer --> 99.13%
start.time <- Sys.time()
history2 <- per2 %>% fit(
  matrix_train_musk, train_musk_y,
  epochs = 100, batch_size = 128, #epochs: numero de passadas no dataset
  validation_split = 0.2
)
per_time2 <- Sys.time() - start.time
print(per_time2)

plot(history2)
history2

#Probar el modelo --> 99.29%
per_test2 <- per2 %>% evaluate(matrix_test_musk, test_musk_y)
per_test2
```



Algunas otras conclusiones que tomamos de este modelo es que:

- En nuestro caso, cuanto más grande el número de variables, aunque no siendo muchas de ellas componentes principales, mejor es para la predicción de estos modelos;
- El valor de accuracy, ha ido mejorando a cada iteración (epoch);
- Por otro lado, el loss ha empeorado a cada iteración, aunque hubo un momento que se estabilizó. Y, considerando que la diferencia entre training y validation sets, para loss, ha sido un valor muy pequeño, todavía damos el modelo como bueno.

Enunciado parte 2) Predicción sobre los datos Kuzushiji utilizando:

- Multinom con nnet
- MLP con Keras
- CNN con Keras

Para realizar este ejercicio, hemos decidido no aplicar más de un tipo de pre-procesamiento de los datos de Kuzushiji, aparte del pre-procesamiento facilitado por los profesores a través del campus. El motivo ha sido porque entendemos que todas las 784 variables son significativas para predecir el resultado de las 10 clases.

No hemos obtenido buenos resultados con multinom, NNET, y tampoco con clasificación linear. El mejor resultado que llegamos, y por lo tanto elegimos con el optimo para este ejercicio, ha sido el ID 9, de CNN con Keras. Con el hemos tenido un 99.98% en training, y 93.9% con tests, con la configuración que presentamos en la propia tabla.

Comentar también que hemos tenido un error en el modelo del ID 8 que, desafortunadamente, no hemos conseguido arreglar. Este modelo es lo mismo que el ID 7, pero habíamos añadido regularización por "dropout". Por lo tanto, creemos que la diferencia del resultado del ID 8 no sería tan significativa en relación al ID 7, porque el "dropout" sólo significaría una mejora en el tiempo de ejecución, considerando que se calcula la probabilidad de una neurona no ser utilizada. Así, el tráfico de red disminuye.

ID	Resumen del Modelo	Configuración	Error rate para los Resultados del modelo para cada caracteristica de pre-procesamiento
1	Multinom con nnet	decay = 0 maxit = 100	Training: 100% Test: 35.14%
2	Multinom con nnet	decay = 0.5 maxit = 200	Training: 100% Test: 35.34%
3	NNET con 10 capas ocultas	decay = 1 maxit = 100	Training: 35.66% Test: 51.37%
4	LDA	Sin Cross Validation	Training: 22.56% Test: 41.1%
5	QDA	Sin Cross Validation	Training: 22.56% Test: 41.55%
			Medido en Accuracy
6	MLP con Keras y 1 layer	Función activación: ReLU Output: Softmax	Training: 93.53% Test: 90.82%
7	MLP con Keras y 4 layers	Función activación: ReLU Output: Softmax	Training: 93.54% Test: 92.53%
8	MLP con Keras y 4 layers	Función activación: ReLU Output: Softmax Dropout	No hemos podido ejecutar debido al error: *Error in py_call_impl(callable, dots$args, dots$keywords) :* *ValueError: Attempt to convert a value (<tensorflow.python.keras.layers.core.Dense object at 0x163246160>) with*

ID	Resumen del Modelo	Configuración	Error rate para los Resultados del modelo para cada caracteristica de pre-procesamiento
			an unsupported type (<class 'tensorflow.python.keras.layers.core.Dense'>) to a Tensor. No hemos conseguido solventarlo.
9	CNN con Keras	optimizer_sgd decay = 0.001 loss='categorical_crossentropy' metrics = "accuracy"	Training: 99.98% Test: 93.9%

A continuación facilitamos el código del modelo del ID 9:

```
library(dplyr)
library(keras)
lenet <- keras_model_sequential() %>%
    layer_conv_2d(filters=20, kernel_size=c(5,5), activation="tanh",
        input_shape=c(28,28,1), padding="same") %>%
    layer_max_pooling_2d(pool_size=c(2,2),strides=c(2,2)) %>%
    layer_conv_2d(filters=50, kernel_size=c(5,5), activation="tanh",
        input_shape=c(28,28,1), padding="same") %>%
    layer_max_pooling_2d(pool_size=c(2,2),strides=c(2,2)) %>%
    layer_flatten() %>%
    layer_dense(units=500, activation="tanh") %>%
    layer_dense(units=10, activation="softmax")

lenet

sgd <- optimizer_sgd(
                    lr=0.05,
                    decay=0.001,
                    momentum=0.8,
                    clipnorm=1.
)
lenet %>% compile(optimizer=sgd,
              loss='categorical_crossentropy',
              metrics = "accuracy"
)

#Entrenar el modelo lenet --> 99.98% de accuracy en los datos de training
lenet %>% fit(
          train$x,
          train$yOneHot,
          batch_size=50,
```

```
          validation_split=0.2,
          epochs=10
)

#Grabar el modelo
lenet %>% save_model_hdf5("lenet-kuzushiji.h5")

#Predict el modelo
lenet <- load_model_hdf5("lenet-kuzushiji.h5")

testnumeric <- as.numeric(test$x)
dim(testnumeric) <- dim(test$x)
pred_prob <- predict(lenet, testnumeric)
head(pred_prob)

predClass <- apply(pred_prob,1,which.max)
predClass <- classString[predClass]
trueClass <- test$yFactor

# Matriz de confusión
(cMatrix <- table(trueClass,predClass))

correctClass <- sum(diag(cMatrix))
total <- sum(cMatrix)
(accuracy <- correctClass/total)
#93.9% de accuracy en test
```

Facilitamos, por fin, el código del ID 7 por si acaso en la corrección de este ejercicio sea posible que compartáis cualquier feedback que pueda ayudarnos a entender probables errores:

```
#### Keras con 4 layers con Softmax y utilizando como función de activaci
ón el ReLU + Drop_out para utilizar menos tiempo de red ####

per <- keras_model_sequential()
per %>% layer_dense(units = 128, activation = "relu",
              input_shape = c(ncol(nnetData_x))) %>%
        layer_dropout(rate = 0.25) #Peta en este paso
        layer_dense(units = 64, activation = "relu") %>%
        layer_dropout(rate = 0.15)
        layer_dense(units = 32, activation = "relu") %>%
        layer_dropout(rate = 0.10)
        layer_dense(units = 16, activation = "relu") %>%
        layer_dropout(rate = 0.05)
        layer_dense(units = levels.class, activation = "softmax")

# Ahora compilamos con validation metric (loss function = binary_crossent
ropy) y un optimizador (accuracy)
per %>% compile(
```

```
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy") #evaluación de lo buena que es la red neuronal
)
```