

DEPARTAMENTO DE PROCESOS Y SISTEMAS

MATERIA: COMPUTACIÓN EMERGENTE

PROFESOR: NICOLAS ARAQUE VOLK

**Proyecto N° 1: Computación Emergente. Informe**

Integrantes:

Catapano, Mario. N° Carnet: 20171110517

Di Gennaro, Massimo. N° Carnet: 20171110123

Pereira, Carlos. N° Carnet: 20161120526

Caracas, Octubre 2019

1. Preprocesamiento de datos: Aquí se deben explicar todas las herramientas y tácticas

que se utilizaron para preprocesar los datos. Variables de entrada al modelo,

normalización, estandarización, etc.

* No se tomó en cuenta los datos de ticket, debido a que el ticket posee un código alfanumérico que no consideramos relevante, ya que la mayor información que se podía recolectar del mismo era únicamente el punto de embarque el cual está presente en un grupo muy pequeño de datos.
* En cuanto a los datos de edad, se realizó un estimado promediando las edades existentes con la finalidad de rellenar aquellas filas de pasajeros sin edad. Para realizar el promedio, se tomó en cuenta el título que posee la persona, ya que existen títulos dentro del dataset que se refieren a personas de baja edad, como por ejemplo, Master.
* En cuanto al nombre tomamos en cuenta el título que se le daba a cada persona, se extrajeron a través de la librería “re” estos 17 distintos títulos que estaban presentes. De esos títulos se agruparon aquellos que solo contenían escasos miembros, quedando 26 tripulantes en la lista de agrupados, que de no ser así hubiese resultado, luego de aplicar el one hot encoding, en 12 columnas adicionales, en donde cada una tendrìa solo unos pocos unos, en toda la matriz. Al realizarlo de esta forma, se consiguiò un total de 6 clases, en las que se clasificaron a los tripulantes.
* Para los datos de la cabina, se tomaron aquellos pasajeros que poseían datos, y se asignó un valor de cero en la matriz para los pasajeros que no tenían datos. Con los datos extraídos, se procedió a realizar un one hot encoding, que dió como resultado un total de 8 columnas adicionales.
* Se arreglaron todos los datos que eran alfanuméricos y se catalogaron, se replantearon, y se le asignó un número a cada dato distinto, para catalogarlos, con la finalidad de realizar el one hot encoding para mejorar y facilitar el aprendizaje.
* Para finalizar se utilizó “StandardScaler” para normalizar los datos. El centrado y el escalado ocurren independientemente en cada función al calcular las estadísticas relevantes de las muestras en el conjunto de capacitación. La media y la desviación estándar se almacenan para ser utilizadas en datos posteriores.

2. Arquitectura del modelo: Aquí se debe explicar la arquitectura utilizada del modelo de Deep Learning. Se deben especificar todos los elementos que hemos visto en clase y algún otro que sea relevante (número de capas, número de neuronas, funciones de activación, regularización, etc)

En la Arquitectura del Modelo, se propuso un total de 7 capas ocultas, donde cada una posee la siguiente cantidad de neuronas: 90 para la primera, 72 para la segunda, 54 para la tercera, 36 para la cuarta, 18 para la quinta, 9 para la sexta, y 4 para la séptima, para un total de 283 neuronas en la capa oculta. Cada neurona en estas 7 capas ocultas posee una función de activación ReLU, excepto en la capa de salida, donde se utiliza la función sigmoide. Para la regularización, se realizó un DropOut en las primeras 6 capas ocultas, con valores P iguales a 0,5 en la primera, 0,4 en la segunda, 0,3 en la tercera, 0,2 en la cuarta, 0,1 en la quinta, y 0,05 en la sexta capa.

def func\_model():

inp = Input(shape=(26,))

x=Dense(90, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(inp)

*#x=Dropout(0.6)(x)*

x=Dense(72, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

x=Dropout(0.6)(x)

x=Dense(54, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

x=Dropout(0.5)(x)

x=Dense(36, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

x=Dropout(0.4)(x)

x=Dense(18, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

*#x=Dropout(0.25)(x)*

x=Dense(9, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

x=Dropout(0.1)(x)

x=Dense(4, activation="relu", kernel\_initializer='glorot\_normal', bias\_initializer='zeros',kernel\_regularizer=regularizers.l2(0.01))(x)

x=Dense(1, activation="sigmoid", kernel\_initializer='glorot\_normal', bias\_initializer='zeros')(x)

model = Model(inputs=inp, outputs=x)

*#sgd = optimizers.SGD(lr=0.0001, clipnorm=1.)*

sgd = optimizers.SGD(lr=0.001, decay=0.001)

model.compile(loss='binary\_crossentropy', optimizer='sgd', metrics=['binary\_accuracy'])

return model

model = func\_model()

print(model.summary())

3. Entrenamiento del modelo: Aquí se debe explicar los elementos de entrenamiento del

modelo (learning rate, epochs, algoritmo de optimización, etc)

Para el entrenamiento, se utilizó un learning rate de valor 0,001, y el learning rate decay es de 0,001. Los epochs para entrenar el modelo son 150. Para la optimización del modelo, se utilizó el algoritmo de optimización Adam, ya que usa momentum para actualizar el paso, lo que evita cambios bruscos en el paso. Esto lo hace muy estable, y además permite calcular un paso global en vez de usar un paso para cada variable.

4. Resultado de modelo: Aquí se debe detallar el resultado del modelo específicamente

en: accuracy en entrenamiento y validación, resultado en leaderboard de Kaggle,

tiempo de ejecución del notebook y cualquier otro indicador que sea relevante

Los resultados obtenidos en el accuracy del modelo fueron los siguientes: para el accuracy de entrenamiento se obtuvo un total de 0.88(AUC), para el accuracy de validación fue de 0.84(AUC), sin embargo, en algunas versiones del proyecto estas alcanzaron valores de 0.90 y 0.93, respectivamente. Para el accuracy del test se obtuvo un resultado de 0.7925. El resultado obtenido, nos sitúa en el leaderboard de Kaggle en la posición 2096 de 12524. El tiempo de ejecución del notebook fue de 23.1 segundos, en la última versión. Sin embargo, para tiempo de ejecución, de forma general, se obtuvieron valores entre 22 y 55 segundos. Según los resultados de Keras, el training accuracy dio como resultado 0.829 Y, por último, el validation Accuracy fue de 0.800, ambos de estos accuracy son de la ultima version mas no de la versión que obtuvo el mejor resultado en el test set.

5. Link al Kernel de Kaggle, este debe hacerse público.

<https://www.kaggle.com/mariocatapano/proyecto-1-titanic>