Entrega 3 Módulo 2

Daniel Humberto Meneses Ortiz A01747963

Profesor:

Jorge Adolfo Ramírez Uresti

11 de septiembre del 2024

Grupo 101

Inteligencia Artificial Avanzada para la Ciencia de Datos

Análisis y reporte sobre el modelo

Para esta tercera entrega, se eligió el modelo propuesto en la Entrega 2, es decir, una red neuronal utilizando "TensorFlow". Como ya se explicó anteriormente, se trata de una red neuronal que clasifica a las personas como "Obesa" o "No obesa", basándose en dos características: su peso y altura. Para poder trabajar correctamente, se siguieron los lineamientos indicados en clase, desde la extracción de datos hasta la prueba del modelo. El propósito del presente documento es conocer la estructura y analizar el desempeño de esta primera red neuronal, para así proponer posibles regularizaciones que permitan mejorar el modelo.

El conjunto de datos con el que se trabaja es el mismo que se utilizó para la entrega anterior. Consiste de tres columnas: "Peso (kg)", "Altura (m)" y Obesidad. También, está compuesto de 1000 observaciones, las cuales corresponden a cada una de las personas capturadas (Fig. 1). Las primeras dos columnas se explican por sí solas, sin embargo, es necesario mencionar que la última contiene las etiquetas de cada persona, es decir, si esta es obesa o no. La razón por la cual el dataset es apropiado para el modelo, es porque al tratarse de un algoritmo de clasificación, y por ende, de aprendizaje supervisado, se necesitan las etiquetas para saber si la red neuronal está haciendo la clasificación correctamente, tomando como entradas el peso y al altura de la persona. Además, cabe destacar que la proporción de clases está balanceada; por eso se seleccionó este conjunto, y se siguió adelante con la construcción del modelo.

Peso (kg)	Altura (m)	Obesidad
87.45401	1.592566	si
145.0714	1.77095	si
123.1994	1.936473	si
109.8658	1.866112	si
65.60186	1.903281	no
65.59945	1.829392	no
55.80836	1.846138	no
136.6176	1.924598	si
110.1115	1.624834	si
120.8073	1.744712	si
52.05845	1.610605	no
146.991	1.993834	si
133.2443	1.97203	si
71.23391	1.519713	si
68.1825	1.852788	no
68.34045	1.962624	no

Fig. 1. Visualización de una parte del conjunto de datos utilizado.

Una vez importadas las librerías y cargados los datos, lo siguiente es dividir el dataset en los conjuntos de entrenamiento, validación y prueba. Para ello, primero se dividió el conjunto en entrenamiento (70%) y prueba (30%). Después se dividió el conjunto de entrenamiento en entrenamiento (80%) y validación (20%). Esto con la finalidad de entrenar a la red neuronal con el dataset de entrenamiento, validar la clasificación con el conjunto de validación, y hacer las predicciones con el dataset reservado para pruebas. De esta forma se garantiza que las predicciones se hacen con datos nunca antes vistos por el modelo (Fig. 2).

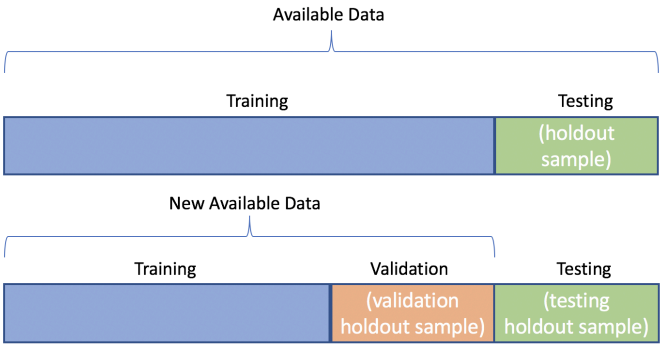


Fig. 2. Explicación de la partición del dataset.

```
pd.DataFrame(X_train, columns=['Peso (kg)', 'Altura (m)'])
```

	Peso (kg)	Altura (m)
0	100.588868	1.756111
1	101.134240	1.740253
2	52.651131	1.950351
3	129.829518	1.621023
4	129.526731	1.998348
...	...	...
555	148.996023	1.552832
556	76.867736	1.794101
557	77.864646	1.933647
558	91.892545	1.828239
559	62.915942	1.762582

560 rows × 2 columns

Fig. 3. Conjunto de entrenamiento.

```
pd.DataFrame(X_val, columns=['Peso (kg)', 'Altura (m)'])
```

	Peso (kg)	Altura (m)
0	81.897563	1.935375
1	63.401523	1.805490
2	53.931214	1.940040
3	60.717201	1.963589
4	126.161962	1.940435
...	...	...
135	122.225257	1.718802
136	57.513778	1.741484
137	94.844552	1.968303
138	79.122914	1.516973
139	87.646337	1.823737

140 rows × 2 columns

Fig. 4. Conjunto de validación.

```
pd.DataFrame(X_test, columns=['Peso (kg)', 'Altura (m)'])
```

	Peso (kg)	Altura (m)
0	88.089086	1.967218
1	131.659944	1.597311
2	96.267979	1.913769
3	85.609673	1.800297
4	145.006197	1.555960
...	...	...
295	72.359584	1.935377
296	77.230733	1.902494
297	85.091501	1.819560
298	136.617615	1.924598
299	74.205527	1.876358

300 rows × 2 columns

Fig. 5. Conjunto de prueba.

Con el dataset dividido, se procede a construir una red neuronal secuencial utilizando la librería “Keras” de “TensorFlow” para realizar una clasificación binaria. La red tiene una estructura simple, contando solo con dos capas: una capa oculta y una capa de salida. En la primera línea, se crea un modelo secuencial, lo que significa que las capas se añaden en secuencia una tras otra. Luego, se añade la primera capa oculta densa, que contiene 4 neuronas y utiliza la función de activación ReLU. Esto es común para capas ocultas en redes neuronales debido a su capacidad para manejar correctamente la no linealidad. La capa de entrada está definida por “input_shape=(X_train.shape[1],)”, lo que indica que el número de entradas es igual al número de columnas en el conjunto de datos de entrenamiento, es decir, dos: peso y altura.

Por su lado, la segunda capa es la capa de salida, la cual cuenta con una sola neurona ya que la tarea es clasificar entre obeso o no obeso. Se usa una función de activación sigmoidal, debido a que la salida necesita ser un valor entre 0 y 1, representando la probabilidad de pertenecer a una de las dos clases. Es necesario mencionar que para compilar el modelo, se utiliza el optimizador Adam, que es eficiente para problemas de clasificación y ajusta dinámicamente la tasa de aprendizaje. Además, la función de pérdida utilizada es “binary_crossentropy”, la cual es adecuada para problemas de clasificación binaria. Por último, la métrica utilizada es “accuracy”, que mide el porcentaje de predicciones correctas, siendo una métrica clave para evaluar el rendimiento del modelo en este tipo de tarea.

El modelo se entrena con los datos “X_train” y “y_train” durante 100 épocas, lo que significa que el conjunto de entrenamiento pasará 100 veces por la red neuronal. Durante el entrenamiento, se valida el modelo en cada época utilizando los datos de validación. Estos no se usan para ajustar los pesos, sino para evaluar el desempeño del modelo en datos no vistos por este. El entrenamiento se realiza con un “batch_size” de 10, lo que significa que el modelo actualiza sus parámetros después de procesar cada grupo de 10 muestras del conjunto de entrenamiento. Finalmente se alcanza una exactitud de 0.9246 y una pérdida de 0.1797 sobre los datos de entrenamiento, y 0.9214 y 0.1894 sobre los de validación (Fig. 6).

```
Epoch 94/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8934 - loss: 0.2124 - val_accuracy: 0.9214 - val_loss: 0.1946
Epoch 95/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9063 - loss: 0.2063 - val_accuracy: 0.9286 - val_loss: 0.1895
Epoch 96/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9078 - loss: 0.1972 - val_accuracy: 0.9286 - val_loss: 0.1885
Epoch 97/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8982 - loss: 0.2014 - val_accuracy: 0.9214 - val_loss: 0.1882
Epoch 98/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9033 - loss: 0.2000 - val_accuracy: 0.9286 - val_loss: 0.1879
Epoch 99/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8991 - loss: 0.2007 - val_accuracy: 0.9286 - val_loss: 0.1909
Epoch 100/100
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9246 - loss: 0.1797 - val_accuracy: 0.9214 - val_loss: 0.1894
```

Fig. 6. Progreso del “accuracy” y “loss” sobre los datos de entrenamiento y validación a lo largo de las épocas.

La Figura 7 muestra el desempeño de la red en términos de pérdida y precisión durante las épocas establecidas. La pérdida tanto en el conjunto de entrenamiento como en el de validación disminuye rápidamente en las primeras épocas y se estabiliza cerca de cero, lo que indica que el modelo aprende apropiadamente. En cuanto a la precisión, ambas curvas aumentan rápidamente y se estabilizan alrededor de un valor elevado, cercano a 0.90. Dado que las curvas de pérdida y precisión son muy similares para ambos datasets, se puede concluir que el modelo está generalizando correctamente y no está sufriendo de sobreajuste.

En términos de bias y varianza, el modelo parece tener un bias bajo, ya que logra una alta precisión y una baja pérdida. Esto quiere decir que está capturando correctamente los patrones de los datos. Por otro lado, la varianza también es baja, ya que las curvas de entrenamiento y validación se comportan de manera similar. Esto indica que el modelo no está sobre ajustado a los datos de entrenamiento y por ende, pueden utilizarse para los datos de prueba. En general, el modelo no parece mostrar signos de underfitting ni overfitting, presentando un equilibrio adecuado entre sesgo y varianza.

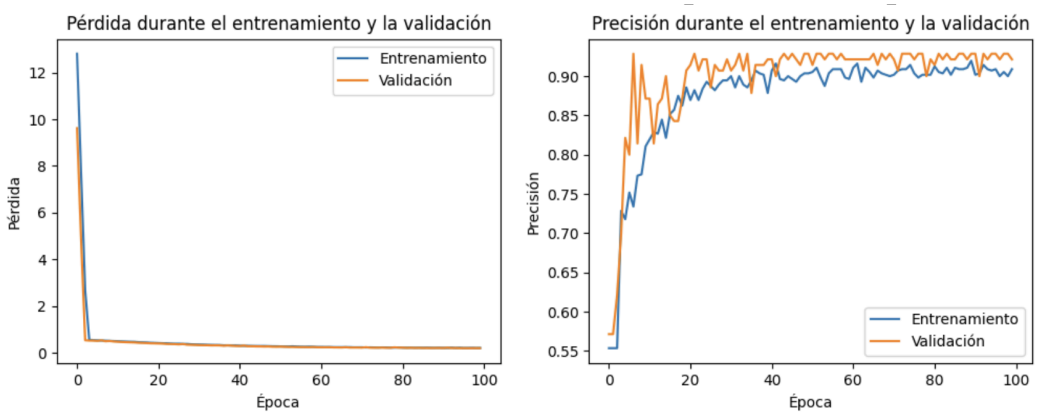


Fig. 7. Gráficas del desempeño de la red en términos de pérdida y precisión durante las épocas establecidas.

La matriz de confusión presentada en la Figura 8 permite visualizar el desempeño del modelo en el conjunto de datos de prueba. En total, el modelo clasificó correctamente 124 casos de la clase 0 o “No Obeso” y 147 casos de la clase 1 o “Obeso”. Hubo casos en los que se equivocó, más específicamente 14 en los que predijo 1 cuando era 0 y 15 en los que predijo 0 cuando era 1. El accuracy obtenido es de 0.9033, lo que indica que el modelo clasifica correctamente en la mayoría de los casos. Además, el recall es de 0.9074, significando que el modelo detecta correctamente los positivos. Finalmente, el F1-score de 0.9102, refleja un buen equilibrio entre ambas métricas. Esto sugiere que el modelo es preciso y también generaliza en los datos de prueba.

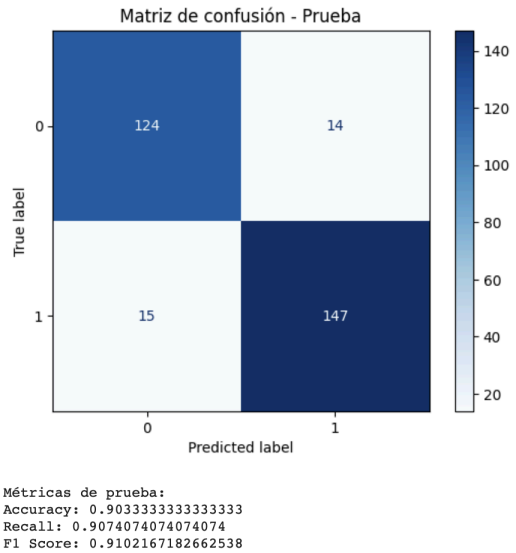


Fig. 8. Matriz de confusión del clasificador sobre los datos de prueba.

Para mejorar el rendimiento y evitar el posible sobreajuste del modelo se pueden emplear técnicas de regularización. Una de las más comunes para las redes neuronales es "Dropout", el cual consiste en desconectar aleatoriamente un porcentaje de las neuronas durante el entrenamiento para evitar que el modelo dependa de combinaciones específicas de neuronas y así alcanzar una mayor generalización. Otra técnica es la Regularización "L2" o "Ridge", que penaliza los grandes valores de los pesos en las conexiones entre neuronas. Esto obliga al modelo a mantener los pesos con valores pequeños, reduciendo en consecuencia la complejidad y evitando que el modelo se sobreajuste a los datos de entrenamiento. "Batch Normalization", aunque no es técnicamente una medida de regularización como tal, ayuda a estabilizar el entrenamiento ya que normaliza las activaciones dentro de cada batch, mejorando la eficiencia del aprendizaje y reduciendo el riesgo de sobreajuste.

Cada una de estas técnicas se puede implementar fácilmente en este modelo de red neuronal. Por su parte, "Dropout" se añade con la función "Dropout(rate=0.5)", donde se indica el porcentaje de neuronas que se desconectan durante cada paso de entrenamiento. "L2" se implementa añadiendo el argumento "kernel_regularizer=l2(0.01)" dentro de las capas, lo que introduce el castigo para los pesos grandes como ya se había explicado. Finalmente, "Batch Normalization" se añade con "BatchNormalization()", que se coloca después de una capa densa para normalizar las activaciones. Los resultados de estas implementaciones se presentan a continuación.

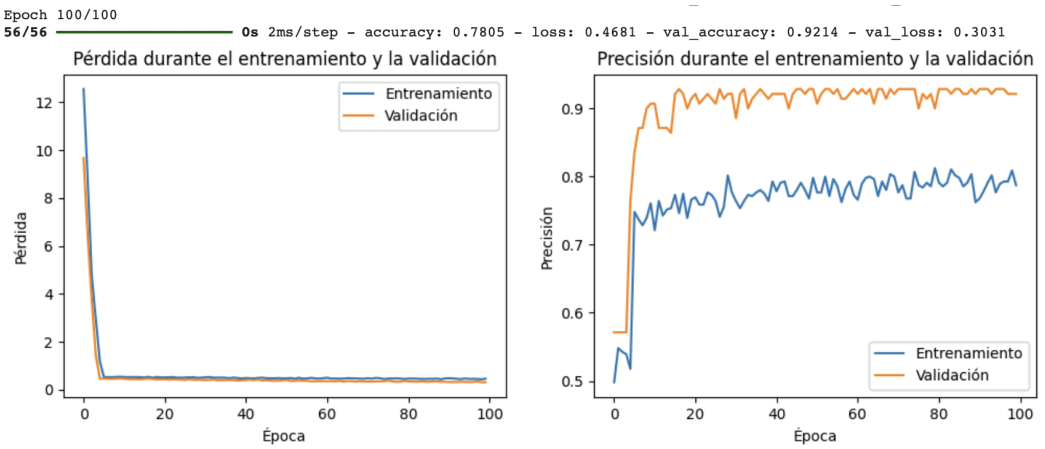


Fig. 9. Gráficas del desempeño de la red con “Dropout” en términos de pérdida y precisión durante las épocas establecidas.

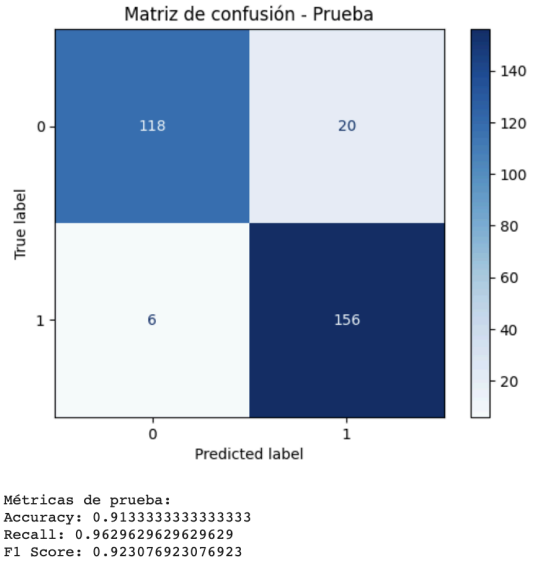


Fig. 10. Matriz de confusión del clasificador con “Dropout” sobre los datos de prueba.

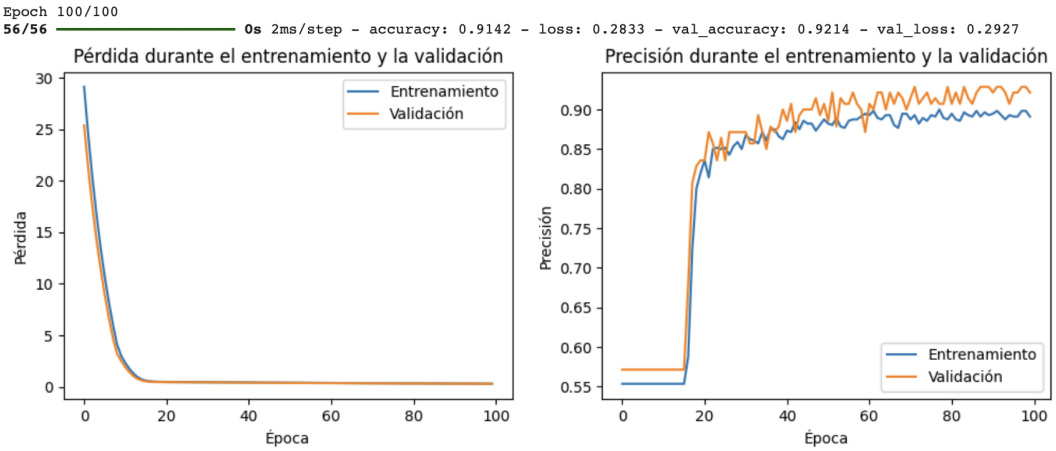
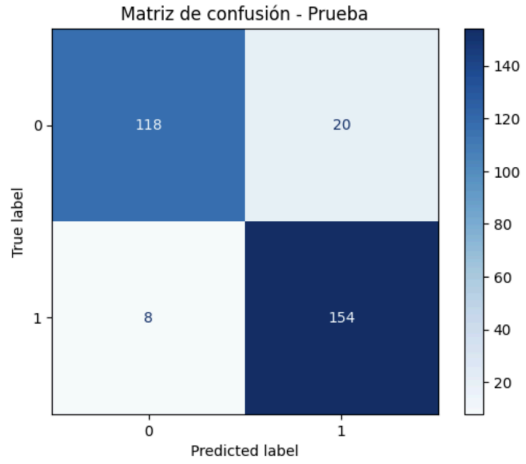


Fig. 11. Gráficas del desempeño de la red con “L2” en términos de pérdida y precisión durante las épocas establecidas.




```
Métricas de prueba:
Accuracy: 0.9066666666666666
Recall: 0.9506172839506173
F1 Score: 0.9166666666666666
```

Fig. 12. Matriz de confusión del clasificador con “L2” sobre los datos de prueba.

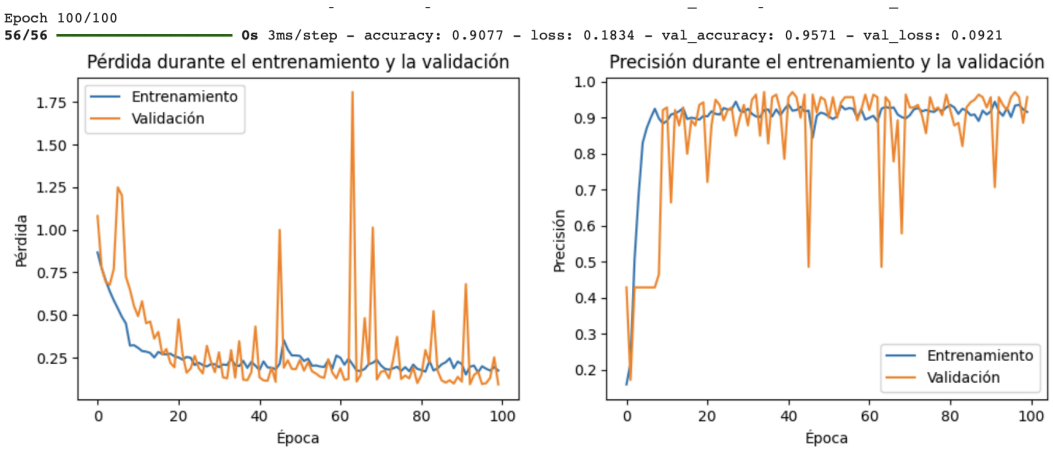
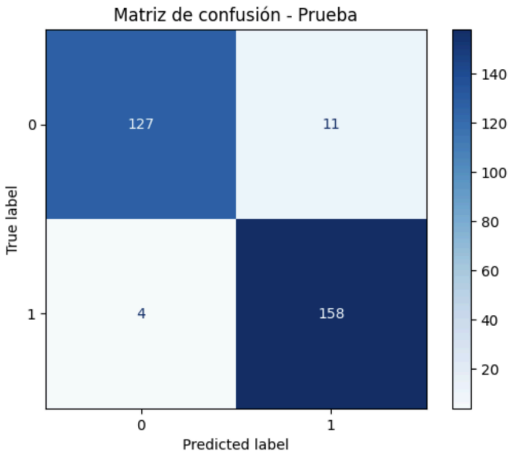


Fig. 13. Gráficas del desempeño de la red con “Batch Normalization” en términos de pérdida y precisión durante las épocas establecidas.




```
Métricas de prueba:
Accuracy: 0.95
Recall: 0.9753086419753086
F1 Score: 0.9546827794561933
```

Fig. 14. Matriz de confusión del clasificador con “Batch Normalization” sobre los datos de prueba.

Al comparar los resultados del modelo original con los tres modelos con alguna medida de regularización, se observa que las técnicas aplicadas mejoran la capacidad de generalización del modelo. Las gráficas de pérdida y precisión muestran que, si bien el modelo original ya presentaba un buen desempeño, los modelos regularizados lograron mantener una tendencia de aprendizaje más estable, particularmente en las últimas épocas. Esto redujo considerablemente la posibilidad de sobreajuste. Particularmente, la red neuronal con "Dropout" mostró un ligero aumento en la estabilidad de las curvas de precisión, mientras que por su parte, "L2" ayudó a controlar los pesos excesivos en la red. Aunado a esto, "Batch Normalization" logró reducir la pérdida en los datos de validación considerablemente, al igual que aumentar el "accuracy" en este mismo dataset.

En cuanto a las matrices de confusión, los modelos con regularización muestran una ligera mejora en la clasificación correcta de las muestras. Presentan menos falsos positivos y falsos negativos en comparación con el modelo original, aumentando en consecuencia el accuracy de las redes en cuestión. Esto es especialmente visible en el primer y tercer modelo, donde se logra un balance más adecuado entre sensibilidad y precisión, evidenciado por el aumento del F1-score. En resumen, las técnicas de regularización no solo evitaron el sobreajuste, sino que también mejoraron la robustez del modelo en los datos de prueba, logrando un mejor rendimiento en términos de generalización.

Referencias

Autor desconocido. (2019, Agosto 20). *Clarification on train, test and val, and how to use/implement it*. Data Science Stack Exchange. https://datascience.stackexchange.com/questions/61467/clarification-on-train-test-and-val-and-how-to-use-implement-it