Módulo 2 Análisis y Reporte sobre el desempeño del modelo

Ana Martínez Barbosa
ITESM CEM

El dataset elegido consta de 500 entradas y 4 features (“altura”, “peso”, “género” y “indice”. El objetivo del modelo era hacer una clasificacion, por ende se decidió transformar la columna de “índice” en una columna booleana para clasificar si la persona tenía obesidad o no. El threshold que se usó fue que el indice fuera mayor a 4 para ser considerado como clasificación.

El dataset es apropiado para el algoritmo ya que, al tratarse de una red neuronal, se pueden hacer tanto regresiónes como clasificaciones. En este caso es una clasificación binaria y en una red neuronal se pueden adaptar ciertos parámetros y especificiar si se trata de una clasificación binaria a diferencia de una multiclase.

Además, el data set eprmite poder entrenar el modelo y hace rmodificaciones de hiperparámetros sin la necesidad de ser edmasido compeloo. No requiere much alimpieza, más que algunas ciertas modificaciones, que facilitan el enfoque principal del proyecto, el cula es desarrollar el modelo. Por su misma simplicidad, permite correr diferentes modelos y experimentar sin la necesidad de un gran consumo de memoria.

I. Separación modelo

Para la modelación del modelo, se dividió el conjunto de datos utilizando la libreria pertinente de sklearn en un conjunto de test (30% de los datos) y otro conjunto de entrenamiento (70% datos).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figura 1. Fragmento de código de separación de datos de prueba y entrenamiento.

```
[ ]  len(X_test)
     150
[ ]  len(X_train)
     350
```

Figura 2.Cantidad de datos pertenecientes a cada conjunto de datos.

Posteriomente la misma red neuronal separa del conjunto de entrenamiento, un conjunto más, el de validación, el cuál representa el 20% de los datos de entrenamiento. Así, verifica el modelo internamente durante su entrenamiento permitiendo que se ajuste correctamente y se hagan las modificaciones a los hiperparámetros pertinentes. Una vez realizado el modelo, se pueden hacer pruebas con el conjunto de “test” que nunca ha visto el modelo.

```
model.fit(X_train, y_train, epochs=10, validation_split=0.2, verbose=2)
```

Figura 3. Fragmento de código que separa un set de los datos de entrenamiento en validación.

II. Bias/sesgo del modelo

Para poder saber si el modelo presenta sesgo se decidió utilizar el mismo modelo pero entrenarlo con diferente split de datos para observar si era acertado cada vez que se corría.

Por lo tanto, se hicieron 3 corridas con diferentes random_seeds y se graficaron los resultados que predijo el modelo vs los resultados correctos.

Como se aprecia a continuación, en el primer random_state (42), solo hubo un error en la predicción:

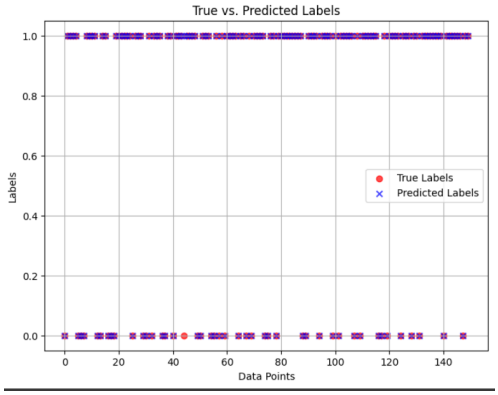

Figura 4. True lables vs predicted random_state = 42

```
accuracy:  0.9866666666666667
```

Figura 5. Accuracy random_state = 42

El segundo intento, con random_state (2), no tuvo ningun error en la predicción:

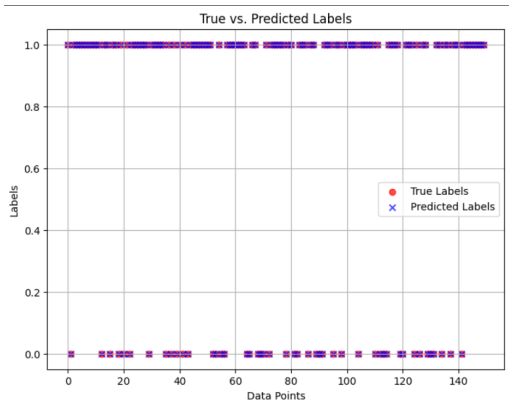

Figura 6. True lables vs predicted

```
accuracy:  1.0
```

Figura 7. Accuracy random_state = 2

Mientras que en el último intento de random_state(20) se obtuvo la mayor cantidad de inaciertos.

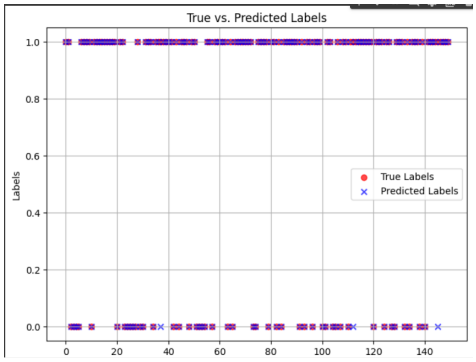

Figura 8. True lables vs predicted

Figura 9. .Accuracy random_state = 20

Además, verificando sus valores de la métrica “accuracy”, los 3 modelos presentan un valor bastante alto, incluso uno que es igual a 1.

A partir de estos resultados se puede apreciar que el modelo presenta un bajo grado de sesgo y que las predicciones son casi en su totalidad acertadas.

III. Separación modelo

Como la varianza mide que tan sensible es el modelo a pequeños cambios en los datos, se decidió evaluar la exactitud del modelo en un set de test diferente.

Por ende divide mi set de datos del conjunto de test en 3 subdivisiones de 50 entradas cada una.

Posteriomente se hacen las predicciones con dichos subsets y de esta manera se obtienen 3 graficas para diferentes datos de test.

Así se puede observar que en los tres intentos, el modelo tiene muy baja varianza ya que sus resultados corresponden con lo que deberian de ser a pesar de los cambios al probarlos.

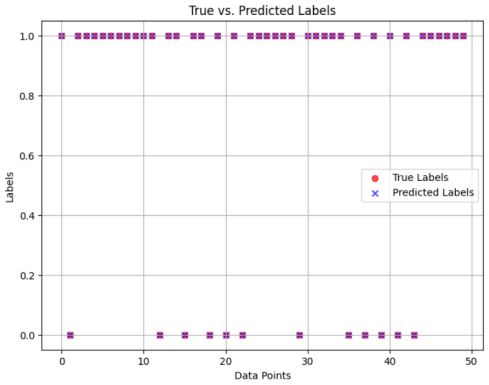

Figura 10: True lables vs predicted

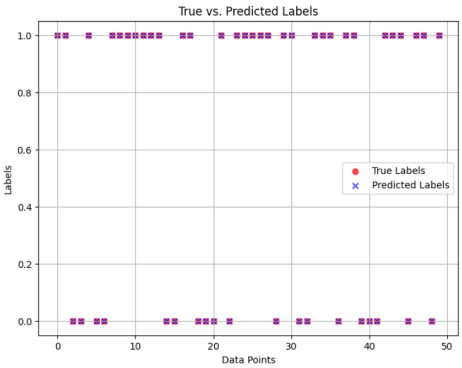

Figura 11: True lables vs predicted

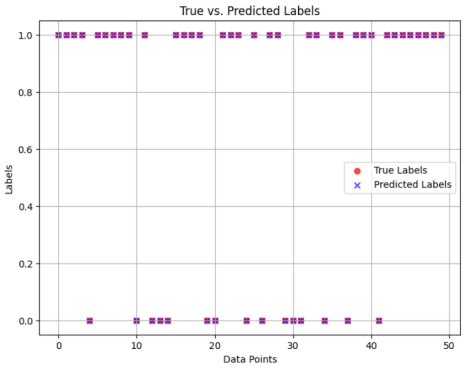

Figura 12: True lables vs predicted

IV. Overfititng vs Underfitting

A partir de las pruebas anteriores, se puede concluir que no posee underfitting el modelo ya que generaliza ben y obtiene muy buenas predicciones y métricas, como se puede ver a continuación.

A partir de la matriz de confusión podemos observar que tiene solamente 4 errores o falsos positivos.

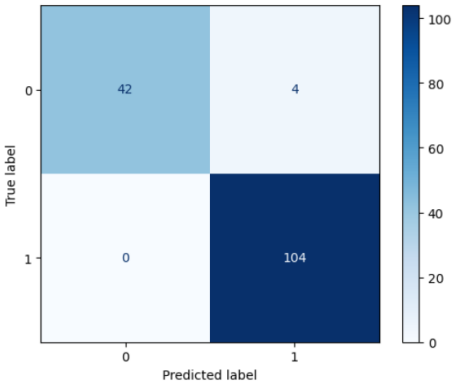

Figura 13: Matriz de confusión

También podemos ver que las métricas de accuracy, precision, recall, F1-score y ROC AUC son bastante altas sin llegar a presentarse demaisdoas "1" que pudieran sugerir overfitting. Solamente en el caso de Recall se podría ver un ajuste "perfecto" que podría llamar la atención.

	Random State	Accuracy	Precision	Recall	F1-Score	ROC AUC
0	42.0	0.973333	0.962963	1.0	0.981132	0.956522

Figura 14: Metrics for model random_state = 42

Otra manera de saber si existe overfitting es haciendo una gráfica comparativa de entrenamiento contra validación.

Figura 15: Training and validation vs epochs

Debido a la escala, puede ser dificil de apreciar que son muy cercanas entre ellas. Podemos ver que en los últimos epochs el validation accuracy alcanza un plateau, lo cual podría indicar overfitting.

Finalmente, clasificaría al modelo con un pequeño grado de overfitting, por lo cuál sería una buna técnica implementar la regularización para ver sus mejoras.

V. Regularización

El objetivo es lograr un equilibrio entre el sesgo y la varianza para obtener un modelo que se generalice bien a datos nuevos.

Por ende, para visualizar qué tanto podría mejorar el modelo, se decidió implementar 2 técnicas de regularización, L1 y L2.

Se esperaba obtener mejores resultados con este ajuste ya que la regularización ayuda a prevenir el sobreajuste, que es algo que podría llegar a ocurrir en nuestro modelo.

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=22, activation='relu', kernel_regularizer=l1(0.05),input_shape=(X_train.shape[1],)))
```

Figura 16: Training and validation vs epochs

Como se puede observar en la métricas, mientras que "recall" se mantuvo en un 1, las demás métricas mejoraron de manera considerable y se encuentran muy cercanas a 1.

	Random State	Accuracy	Precision	Recall	F1-Score	ROC AUC
0	42.0	0.993333	0.990476	1.0	0.995215	0.98913

Figura 17: Metrics for model L1

Así mismo, solo presentó un falso positivo.

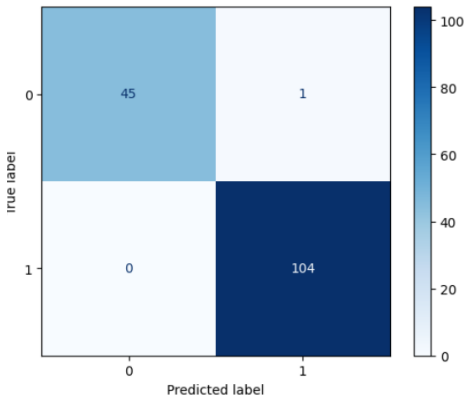

Figura 18: Training and validation vs epochs

Por otro lado, al implementar la regularización L2, se puede observar que "recall" baja ligeramente, mientras que "precision" es un 1. En los demás casos las métricas mejoraron de manera considerable y se encuentran muy cercanas a 1.

	Random State	Accuracy	Precision	Recall	F1-Score	ROC AUC
0	42.0	0.986667	1.0	0.980769	0.990291	0.990385

Figura 19: Metrics for model L1

Por el lado de la matriz de confusión, en este caso, se reducen por completo los falsos negativos, pero aumentan los falsos positivos.

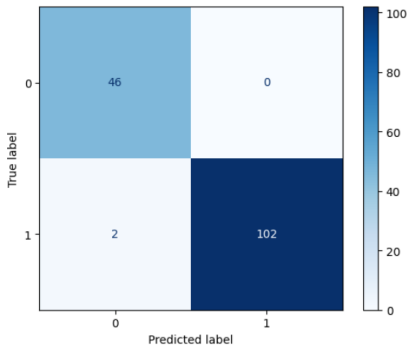

Figura 20: Training and validation vs epochs

De manera global, los resultados más altos los presenta la regularización L1. Sin embargo, la decisión de cuál de los dos métodos usar dependería del objetivo final de clasificación y de si lo más importantes es reducir falsos negativos o falsos positivos.