Reporte de Análisis y Mejora de Modelo de Predicción de BMI

Iván L. Hernández Buda

11 de septiembre de 2023

Resumen

Este reporte presenta el análisis del desempeño del modelo de redes neuronales para predicción del índice de masa corporal (BMI) con un database de altura y peso. El objetivo de este documento es mostrar la evaluación del grado de sesgo, varianza y ajuste del modelo, para posteriormente, mejorar su desempeño mediante técnicas de regularización y ajuste de parámetros. Se utilizó una red neuronal profunda para este propósito y se evaluaron distintas cantidades de neuronas en la capa oculta para determinar su impacto en el rendimiento del modelo y encontrar un punto óptimo entre sesgo y varianza.

1. Introducción

La predicción del índice de masa corporal es importante para la salud y el bienestar de las personas. Este informe se enfoca en analizar y mejorar el rendimiento del modelo de red neuronal basado en dato de altura y peso. El análisis incluye la evaluación del sesgo, de la varianza y el ajuste del modelo, todo esto para comprender su desempeño a lo largo del cambio en sus configuraciones. Luego, se aplican técnicas de ajuste de parámetros y regularización para mejorar su precisión.

Dataset Utilizado: Nombre de archivo: 'bmi.csv' Variables presentes: Id, Gender, Heigth, Weight Número de registros: 500 Fuente: https://www.kaggle.com/datasets/yasserh/bmidataset

Normatividad del Dataset: La normativa asociada al tipo de datos utilizados para 'Machine Learning for body-mass index prediction' son de: CC0 1.0 Universal (CC0 1.0). Public Domain Dedication.

2. Desarrollo

Carga del dataset y feature engineering

Primeramente, se preraró el framework que se usará para la construcción de nuestro modelo de red neuronal con las librerías como sklearn, tensorflow y keras. Después, extraemos los datos del archivo .csv y se cambió a variable dummie la variable 'Gender'. Por último, se separaron los datos como 'X' (input: Gender, Height, Weight) y 'y' (output: Index).

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import make_scorer, mean_squared_error
from sklearn.model_selection import KFold
```

```
# Load the data
data = pd.read_csv("bmi.csv", sep=",")
gender_dummie = pd.get_dummies(data['Gender'],
                               drop_first=True)
gender = gender_dummie.values.reshape(-1)
height = data['Height']
weight = data['Weight']
bmi = data['Index']

random_state = 53
np.random.seed(random_state)
noise = np.random.normal(0, 1, len(data))
data['Height'] = data['Height'] + noise/100
data['Weight'] = data['Weight'] + noise

X = np.array([gender, height, weight]).T
y = bmi
```

Gender	Height	Weight	Index
Male	1.74	96	31.708284
Male	1.89	87	24.355421
Female	1.85	110	32.140248
Female	1.95	104	27.350427
Male	1.49	61	27.476240
...	...	...	...

Cuadro 1: Dataframe de masa de índice corporal

Separación y visualización de los datos

Se dividió el conjunto de datos en conjuntos de entrenamiento y prueba con una proporción de 80/20, respectivamente. Esto permitió evaluar el rendimiento del modelo en datos no vistos. El conjunto de validación no se utilizó en este análisis, ya que se centró en la evaluación de sesgo, varianza y ajuste. Después, se visualizan los datos en una gráfica 3D donde de manera clara, se perciben los datos de prueba y los de entrenamiento.

```
# Data segmentation
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=53)

# Visualize data
print(data.head())
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_train[:,1], X_train[:,2], y_train,
           label='Train data')
ax.scatter(X_test[:,1], X_test[:,2], y_test,
           label='Test data')
```

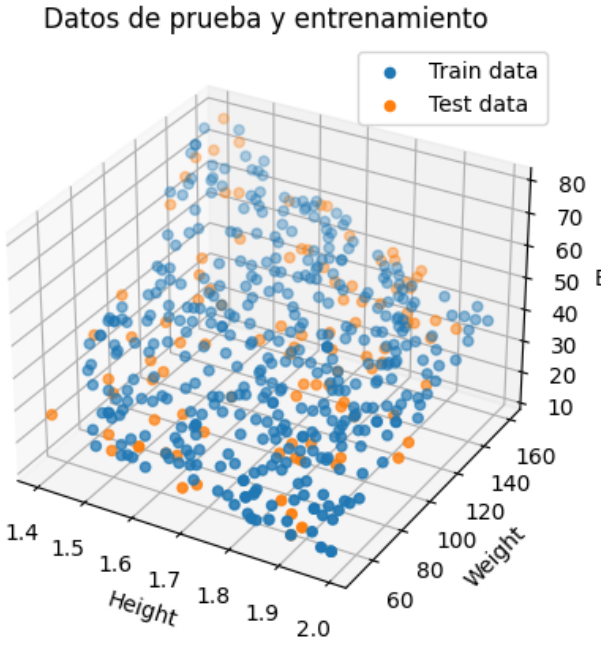

Figura 1: Visualización de datos de prueba y entrenamiento

```
ax.set_xlabel('Height')
ax.set_ylabel('Weight')
ax.set_zlabel('BMI')
ax.legend()
plt.show()
```

Creación de red neuronal

La red neuronal implementada consiste en 3 entradas, una cantidad de neuronales variable para al primera capa oculta, 3 capas ocultas con 4 neuronas cada una, y una salida.

```
# Create the neural network model
def create_model(n_neurons):
    model = MLPRegressor(hidden_layer_sizes=(n_neurons, 2*n_neurons,
    2*n_neurons, n_neurons), activation='relu', random_state=42)
    return model
```

Diagnóstico de sesgo y varianza

El sesgo del modelo se evaluó observando el desempeño en los conjuntos de entrenamiento y los datos de entrenamiento predichas por el modelo. Si el error en el conjunto de entrenamiento es significativamente menor que en el conjunto de prueba, se considera que el modelo tiene un sesgo bajo.

La varianza se evaluó observando la diferencia entre el error en los conjuntos de prueba y los datos de prueba predichas por el modelo. Si esta diferencia es alta, indica una alta varianza, lo que significa que el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a datos no vistos.

En este caso, al realizar el análisis en la figura 2, podemos observar que al inicio, la red neuronal tipo densidad neuronal tipo (25, 50, 50, 25), presenta un sesgo alto, ya que los datos de prueba se ajustan mejor a los datos de entrenamiento pero no hay una diferencia significativa; esto se traduce a un **underfitting**. Por el otro lado, al llegar a la configuración (300, 600, 600, 300), los datos de entrenamiento se ajustan ligeramente mejor a los datos de prueba, lo que se traduce a una varianza media; esto es un ligero **overfitting**.

```
train_mse = []
test_mse = []

for n_neurons in np.arange(1, 600, 25):
  model = create_model(n_neurons)
  model.fit(X_train, y_train)

  y_model_train = model.predict(X_train)
  y_model_test = model.predict(X_test)

  train_mse.append(mean_squared_error(y_model_train, y_train))
  test_mse.append(mean_squared_error(y_test, y_model_test))

plt.figure()
plt.plot(np.arange(1, 600, 25), train_mse,
         label='Training Error', marker='.')
plt.plot(np.arange(1, 600, 25), test_mse,
         label='Test Error', marker='.')
plt.xlabel('Neural density in hidden layer')
plt.ylabel('Mean Squared Error')
plt.title('MSE vs Neural density')
plt.ylim([40,100])
plt.xlim([0,325])
plt.legend()
plt.grid()
plt.show()
```

Diagnóstico del Ajuste del Modelo

El ajuste del modelo se analizó considerando diferentes cantidades de neuronas en la capa oculta de la red neuronal usando la técnica de GridSearch. Se entrenaron modelos con un número creciente de neuronas y se registraron los errores de entrenamiento y prueba para cada uno. Esto permitió determinar si el modelo estaba subajustado (underfitting) o sobreajustado (overfitting) a los datos y cuál era la mejor combinación de parámetros.

```
# Define the hyperparameters and their possible values
```

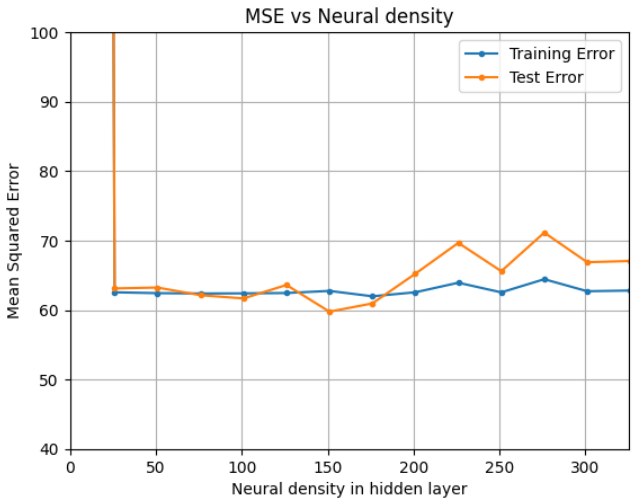

Figura 2: Visualización de sesgo y varianza del modelo

```
layer_sizes = []
for n_neurons in np.arange(1, 600, 25):
  layer_sizes.append((n_neurons, 2*n_neurons, 2*n_neurons, n_neurons))

param_grid = {
    'hidden_layer_sizes': layer_sizes,
    'activation': ['relu'],
    'solver': ['adam'],
}

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=3)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

3. Resultados

Después del implementar la técnica de GridSearch para el ajuste del modelo, se encontró que la mejor configuración que nuestra red neuronal no recaiga ni en underfitting ni overfitting es (151, 302, 302, 151), con una función de activación ReLu (no lineal), un optimizador Adam, y esto brindó un error cuadrado medio de 59.765, el cuál es uno de los errores más bajos que se pueden observar en figura 2.

Métrica	**Valor**
Mean Squared Error on Test Set	59.765
Best Hyperparameters	
Activation	relu
Hidden Layer Sizes	(151, 302, 302, 151)
Solver	adam

Cuadro 2: Resultados del modelo

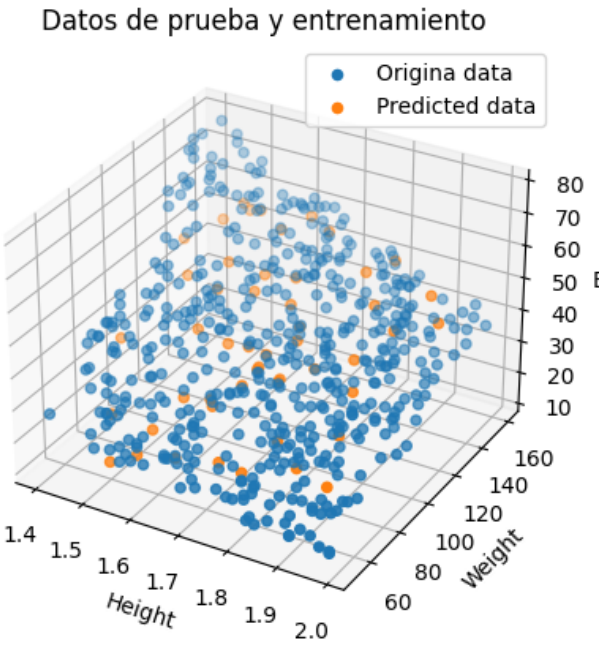

Figura 3: Visualización de predicciones

4. Conclusiones

Sesgo y varianza: Inicialmente se observó que el modelo padecía de un ligero underfitting, debido a que su configuración de red neuronal presentaba un bajo número de neuronas (baja complejidad). A medida que se aumentaba el número de neuronas (alta complejidad), la varianza, en promedio, aumentaba, y por el otro lado, el sesgo disminuía, lo cual indicaba un ligero overfitting. Esto resalta la importancia de encontrar un equilibrio entre sesgo y varianza para lograr un buen ajuste del modelo encontrando el punto de complejidad media.

Ajuste del modelo: La aplicación de GridSearch para ajustar los hiperparámetros de la red neuronal resultó en una mejora en el rendimiento del modelo. La combinación óptima de hiperparámetros encontrada fue una configuración de capas ocultas de (151, 302, 302, 151) con activación ReLU y optimizador Adam. Esto llevó a una reducción significativa en el error cuadrado medio en

el conjunto de prueba. Sin embargo, cabe resaltar que el proceso iterativo para encontrar de manera más precisa la mejor configuración de hiperparámetros requiere un alto poder computacional, pero también, un tiempo de espera elevado.

Mejora del Modelo: En orden de mejorar reducir lo más posible el error cuadrado medio sin recaer en underfitting u overfitting, se aplicaron los hiperparámetros encontrados con el GridSearch, lo cual logró que en el conjunto de prueba, el error se redujera considerablemente, lo que indica una mayor capacidad del modelo para generalizar a datos no vistos.

Referencias

[1] TensorFlow Authors. (2021). TensorFlow: A machine learning framework for everyone. `https://www.tensorflow.org/`

[2] Keras Contributors. (2021). Keras API reference / Sequential. `https://www.tensorflow.org/api_docs/python/tf/keras/Sequential`

[3] Keras Contributors. (2021). Keras API reference / Layers / Dense. `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense`

[4] scikit-learn Developers. (2021). scikit-learn: Machine Learning in Python. `https://scikit-learn.org/stable/`

[5] scikit-learn Developers. (2021). Model evaluation: quantifying the quality of predictions. `https://scikit-learn.org/stable/modules/model_evaluation.html`