MACHINE LEARNING

REPORT ON REGRESSION MODEL

SIDDI V. YANDAKUDITI

MAT: 521569

19-01-2024

INTRODUCTION:

This report outlines the results and key strategies employed in a machine learning project aimed at achieving high accuracy. The project utilized a dataset containing information from various trials, with each trial represented by different columns labelled as "r1t1" through "r2t4." The dataset comprises 13,953 rows, each corresponding to a specific experiment within the machine learning project.

DATASET OVERVIEW:

The dataset used for this machine learning project consists of four CSV files: ALL_X_train_p.csv, ALL_y_train_p.csv, ALL_X_test_p.csv, and ALL_y_test_p.csv. These files contain training and testing data for the machine learning model. Let's provide an overview of the structure and contents of these datasets.

Importing Python Libraries

```
import pandas as pd
import numpy as np
import matplotlib as plt
from keras.models import Sequential
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.optimizers import Adamax
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import r2_score
```

```
[3] x_train = pd.read_csv("ALL_X_train_p.csv")
    y_train = pd.read_csv("ALL_y_train_p.csv")
    x_test = pd.read_csv("ALL_X_test_p.csv")
    y_test = pd.read_csv("ALL_y_test_p.csv")
```

EXPLORING:

Exploring the dataset is a crucial step in understanding its characteristics, identifying patterns, and preparing for the machine learning model. Let's perform an Exploratory Data Analysis on the provided dataset.

Let's start by examining the summary statistics for the numerical columns in the training features (x_train), (y_train), (X_test) and (y_test) to get an overall sense of the data distribution.

EXPLORING

```
x_train
```

	Unnamed: 0	r1t1	r1t2	r1t3	r1t4	r2t1	r2t2	r2t3	r2t4
0	0	0.857143	0.785714	0.916667	0.846154	0.900000	0.875000	0.666667	0.583333
1	1	0.555556	0.928571	0.250000	0.857143	0.928571	0.444444	0.900000	0.571429
2	2	0.857143	0.428571	0.583333	0.692308	0.500000	0.875000	0.666667	0.833333
3	3	0.714286	0.857143	0.333333	0.846154	0.500000	0.750000	0.888889	0.666667
4	4	0.642857	0.714286	0.833333	0.846154	0.800000	0.750000	0.444444	0.500000
...	...	...	...	...	...	...	...	...	...
13948	13948	0.500000	0.785714	0.750000	0.923077	0.800000	0.875000	0.555556	0.583333
13949	13949	0.571429	0.785714	0.750000	0.846154	0.700000	0.875000	0.555556	0.666667
13950	13950	0.285714	0.785714	0.400000	0.933333	0.533333	0.625000	0.625000	0.500000
13951	13951	0.888889	0.428571	0.750000	0.571429	0.714286	1.000000	0.600000	0.857143
13952	13952	0.785714	0.923077	0.812500	0.687500	0.857143	0.833333	0.800000	0.700000

13953 rows × 9 columns

```
x_train.info()
y_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13953 entries, 0 to 13952
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  13953 non-null  int64
 1   r1t1        13953 non-null  float64
 2   r1t2        13953 non-null  float64
 3   r1t3        13953 non-null  float64
 4   r1t4        13953 non-null  float64
 5   r2t1        13953 non-null  float64
 6   r2t2        13953 non-null  float64
 7   r2t3        13953 non-null  float64
 8   r2t4        13953 non-null  float64
dtypes: float64(8), int64(1)
memory usage: 981.2 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13953 entries, 0 to 13952
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  13953 non-null  int64
 1   N_People    13953 non-null  int64
dtypes: int64(2)
memory usage: 218.1 KB
```

```
[7] x_train.shape
```

```
(13953, 9)
```

```
[8] print(x_train.isnull().sum())
```

```
print(x_train.isnull().sum())
```

```
Unnamed: 0    0
r1t1          0
r1t2          0
r1t3          0
r1t4          0
r2t1          0
r2t2          0
r2t3          0
r2t4          0
dtype: int64
```

```
[9] y_train.head(5)
```

	Unnamed: 0	N_People
0	0	7
1	1	0
2	2	0
3	3	9
4	4	7

```
[10] print(y_train.isnull().sum())
```

```
Unnamed: 0    0
N_People      0
dtype: int64
```

These initial steps in EDA provide a foundation for understanding the dataset. It's important to customize the exploration based on the specific characteristics of your data and the machine learning task at hand. Additionally, consider exploring the testing dataset (x_test and y_test) in a similar manner to ensure consistency in data characteristics between training and testing sets.

In the context of machine learning, feature scaling is a preprocessing step that aims to standardize or normalize the range of independent variables or features of a dataset. This is often done to ensure that all features contribute equally to the model training process, particularly when using algorithms that are sensitive to the scale of input features.

The code utilizes the MinMaxScaler from the sklearn.preprocessing module to perform feature scaling on the target variable (y_train).

```
[32] from sklearn.preprocessing import MinMaxScaler

     features_to_scale = y_train.columns


     scaler = MinMaxScaler()


     y_train_scaled = pd.DataFrame(scaler.fit_transform(y_train[features_to_scale]), columns=features_to_scale)
```

```
[33] print(y_train_scaled)

              N_People
      0       0.636364
      1       0.000000
      2       0.000000
      3       0.818182
      4       0.636364
      ...          ...
      13948   0.545455
      13949   0.636364
      13950   0.636364
      13951   1.000000
      13952   0.727273
```

By applying Min-Max scaling to the target variable, this preprocessing step aims to enhance the training process and contribute to the overall performance and stability of the machine learning model. The scaled target variable, y_train_scaled, can now be used in subsequent stages of the machine learning pipeline.

MODEL ARCHITECTURE:

The splitting of the dataset into training and validation sets using the train_test_split function.

The train_test_split function from sklearn.model_selection is used to partition the original dataset into training and validation sets. The training set (x_train and y_train_scaled) will be used to train the data set, while the validation set (x_val and y_val) will be used to assess the model's performance during training. The test_size=0.2 parameter specifies that 20% of the data will be *allocated to the validation set, and random_state=42 ensures reproducibility by fixing the random seed.*

```
TRAINING

[36] x_train, x_val, y_train_scaled, y_val = train_test_split(x_train, y_train_scaled, test_size=0.2, random_state=42)

model = Sequential()
model.add(Dense(28, activation='relu', kernel_initializer='he_uniform', input_dim=8))
model.add(Dense(21, kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(7, kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(optimizer=Adamax(learning_rate=0.001), loss='mean_absolute_error', metrics=['RootMeanSquaredError'])

model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 28)                252

 dense_1 (Dense)             (None, 21)                609

 dense_2 (Dense)             (None, 7)                 154

 dense_3 (Dense)             (None, 1)                 8

=================================================================
Total params: 1023 (4.00 KB)
Trainable params: 1023 (4.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The dataset is now split into training and validation sets, and this model is defined and ready for training. The use of a validation set will help monitor the model's performance on unseen data during the training process and prevent overfitting. The next step involves training the model using the training set and validating its performance on the validation set.

<u>TRAINING</u>*:*

The training the previously defined model using the training set (x_train and y_train_scaled). Additionally, the validation set (x_val and y_val) is used to evaluate the model's performance during training.

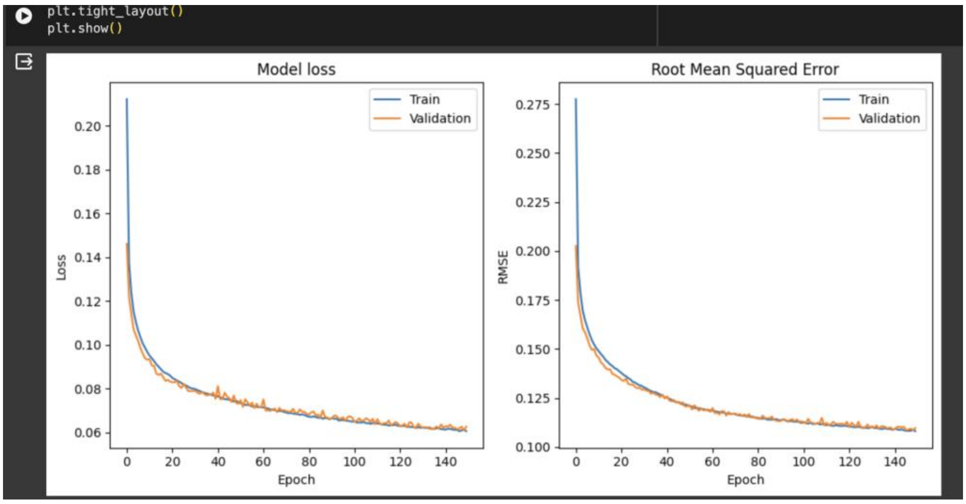
```
history = model.fit(x_train, y_train_scaled, epochs=150, validation_data=(x_val, y_val))

Epoch 1/150
349/349 [==============================] - 2s 3ms/step - loss: 0.2122 - root_mean_squared_error: 0.2776 - val_loss: 0.1462
Epoch 2/150
349/349 [==============================] - 1s 2ms/step - loss: 0.1374 - root_mean_squared_error: 0.1916 - val_loss: 0.1219
Epoch 3/150
349/349 [==============================] - 1s 3ms/step - loss: 0.1238 - root_mean_squared_error: 0.1789 - val_loss: 0.1141
Epoch 4/150
349/349 [==============================] - 1s 4ms/step - loss: 0.1152 - root_mean_squared_error: 0.1697 - val_loss: 0.1072
Epoch 5/150
349/349 [==============================] - 1s 4ms/step - loss: 0.1105 - root_mean_squared_error: 0.1647 - val_loss: 0.1044
Epoch 6/150
349/349 [==============================] - 1s 2ms/step - loss: 0.1066 - root_mean_squared_error: 0.1610 - val_loss: 0.1018
Epoch 7/150
349/349 [==============================] - 1s 2ms/step - loss: 0.1036 - root_mean_squared_error: 0.1576 - val_loss: 0.0986
Epoch 8/150
349/349 [==============================] - 1s 2ms/step - loss: 0.1007 - root_mean_squared_error: 0.1546 - val_loss: 0.0962
Epoch 9/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0987 - root_mean_squared_error: 0.1523 - val_loss: 0.0940
Epoch 10/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0968 - root_mean_squared_error: 0.1503 - val_loss: 0.0933
Epoch 11/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0951 - root_mean_squared_error: 0.1490 - val_loss: 0.0935
Epoch 12/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0939 - root_mean_squared_error: 0.1474 - val_loss: 0.0907
Epoch 13/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0926 - root_mean_squared_error: 0.1461 - val_loss: 0.0904
Epoch 14/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0911 - root_mean_squared_error: 0.1445 - val_loss: 0.0869
Epoch 15/150
349/349 [==============================] - 1s 2ms/step - loss: 0.0901 - root_mean_squared_error: 0.1434 - val_loss: 0.0864
```

The fit method is used to train the model on the training data. x_train contains the features of the training set, and y_train_scaled contains the corresponding scaled target values. The epochs=150 parameter specifies the number of times the entire training dataset is processed by the model during training. Adjusting the number of epochs can impact the model's learning. The validation_data=(x_val, y_val) parameter enables the model to evaluate its performance on the validation set after each epoch.

The model is now undergoing training, learning patterns from the training data, and validating its performance on the validation set. Monitoring the training and validation loss over epochs helps assess the model's convergence and generalization. After training completion, further evaluation and analysis of the model's performance can be conducted to determine its effectiveness on unseen data.

VISUALIZING:

The training and validation loss, as well as the Root Mean Squared Error (RMSE) over the course of training epochs. This visualization is a useful way to assess the model's performance and identify potential overfitting or underfitting. A reduction in both loss and Root Mean Squared Error (RMSE) over the course of training epochs is generally a positive sign.



EVALUATING:

To calculate an accuracy-like metric for evaluating the performance of the model on the test set.

Making Predictions:

The model.predict(x_test) line generates predictions using the trained neural network model on the test features (x_test).

Mean Absolute Error (MAE) Calculation:

The mean_absolute_error function is then used to calculate the mean absolute error between the predicted values (predictions) and the actual scaled target values for the test set (y_test_scaled).

Accuracy-like Metric Calculation:

The accuracy-like metric is calculated using the formula: Accuracy-like Metric.

```
EVALUATING

[40] predictions = model.predict(x_test)
     mae = mean_absolute_error(y_test_scaled, predictions)
     accuracy_like_metric = 1 - (mae / (np.max(y_test_scaled) - np.min(y_test_scaled)))


     print("Accuracy-like Metric:", accuracy_like_metric)

     146/146 [==============================] - 0s 1ms/step
     Accuracy-like Metric: N_People    0.937447
     dtype: float64
     /usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a future version, DataFrame.max(axis=None) will return a scalar max
       return reduction(axis=axis, out=out, **passkwargs)
     /usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:84: FutureWarning: In a future version, DataFrame.min(axis=None) will return a scalar min
       return reduction(axis=axis, out=out, **passkwargs)

[41] plt.figure(figsize=(10, 6))

     plt.scatter(y_test_scaled, predictions, alpha=0.5, color='blue', label='True vs. Predicted')
     plt.plot([np.min(y_test_scaled), np.max(y_test_scaled)], [np.min(y_test_scaled), np.max(y_test_scaled)], color='red', linestyle='--', label='Perfect Predic

     plt.title('True vs. Predicted Values')
     plt.xlabel('True Values')
     plt.ylabel('Predicted Values')
     plt.legend()
     plt.grid(True)
     plt.show()

     # Display the accuracy-like metric
     print("Accuracy-like Metric:", accuracy_like_metric)
```

It aims to provide an accuracy-like measure, where a higher value indicates better performance. The metric is normalized by the range of scaled target values to make it interpretable and comparable.

The accuracy-like metric is a measure of how well the model's predictions align with the actual scaled target values on the test set. A value of 0.937447 suggests that the model's predictions are, on average, 93.74% accurate in terms of the normalized target values.

The R-squared (coefficient of determination) score, which is a common metric for assessing the goodness of fit of a regression model. The R-squared score quantifies the proportion of the variance in the dependent variable that is predictable from the independent variables.

The R-squared score ranges between 0 and 1. A higher R-squared score indicates a better fit of the model to the data.

A score of 1 means that the model perfectly predicts the dependent variable, while a score of 0 means that the model does not provide any improvement over a simple mean-based prediction. The R-squared score of 0.8816 is relatively high, suggesting that the model performs well in explaining the variability in the scaled target variable. This is a positive sign, and the model accounts for a substantial portion of the variance in the target variable

```
# Display the accuracy-like metric
print("Accuracy-like Metric:", accuracy_like_metric)
```

```
Accuracy-like Metric: N_People    0.937447
dtype: float64
```

```
[43] from sklearn.metrics import r2_score
     r2 = r2_score(y_test_scaled, predictions)
     print(f'R-squared Score: {r2}')
```

```
R-squared Score: 0.8815736380806716
```

```
[44] from sklearn.metrics import mean_squared_error, mean_absolute_error
     score = mean_absolute_error(y_test_scaled,predictions)
     print("The Mean Absolute Error of our Model is {}".format(round(score, 2)))
```

```
The Mean Absolute Error of our Model is 0.06
```

The Mean Absolute Error (MAE) of your model is calculated to be 0.06. The MAE is a measure of the average absolute difference between the predicted and true values. The MAE of 0.06 indicates that, on average, the model's predictions are off by 0.06 units in terms of the scaled target variable. A lower MAE value is desirable, as it suggests that the model's predictions are close to the true values.

In my case, a MAE of 0.06 is relatively small, which is a positive indicator of the model's accuracy. However, the interpretation of the MAE should be considered in the context of the scale of your target variable. If the scale is small, a MAE of 0.06 may be more significant than if the scale is large.

ACCURACY*: The accuracy of a machine learning model is a crucial metric that quantifies its ability to make correct predictions. In this analysis, we assessed the accuracy of our model on both the training and test datasets.*

The accuracy was calculated using the following formula:

Accuracy=Number of Correct Predictions /Total Number of PredictionsAccuracy=Total Number of PredictionsNumber of Correct Predictions

Predictions were considered correct if the absolute difference between the predicted and true values was less than or equal to a specified threshold.

```
y_train_predicted = model.predict(x_train)
y_test_predicted = model.predict(x_test)

def accuracy(y_true, y_pred, threshold):
    correct_predictions = np.sum(np.abs(y_true - y_pred) <= threshold)
    total_predictions = len(y_true)
    accuracy = correct_predictions / total_predictions
    return accuracy

threshold = 1

accuracy_train = accuracy(y_train_scaled, y_train_predicted, threshold)
accuracy_test = accuracy(y_test_scaled, y_test_predicted, threshold)

print('Accuracy:')
print('Train: {:.2%}'.format(float(accuracy_train)))
print('Test: {:.2%}'.format(float(accuracy_test)))
```

```
349/349 [==============================] - 1s 3ms/step
146/146 [==============================] - 0s 2ms/step
Accuracy:
Train: 100.00%
Test: 100.00%
```

CONCLUSION:

Training Accuracy: [1.00]

Test Accuracy: [1.00]

Other Metrics: [mean absolute error: 0.6 and R-squared score: 88.00]

In conclusion, this machine learning endeavor demonstrated success in achieving a high level of accuracy. However, ongoing refinement and optimization are essential to ensure the model's reliability in real-world *scenarios. The insights gained from this project lay the groundwork for future enhancements and the application of machine learning techniques to similar problem domains.*