A1: Prediction with Back-Propagation and Linear Regression
Professors: SERGIO GÓMEZ JIMÉNEZ, JORDI DUCH GAVALDÀ
Student: SEBASTIAN-EUGEN BUZDUGAN

Table of Contents

Introduction 2
Objective 2
Deliverables 3
Datasets Analysis 4
Selection and Preprocessing 4
Preprocessing Techniques 7
Implementation 8
Neural Network with Back-Propagation 8
Evaluation Metrics for the Real Estate Dataset 11
Evaluation Metrics for the synthetic dataset 13
Evaluation Metrics for the turbine dataset 14
BP-F – Analysis of the BP-F Dataset Using Neural Network 15
Analysis of the A1-Turbine Dataset Using the Neural Network 15
Analysis of the A1-Synthetic Dataset Using the Neural Network 16
Analysis of the A1 Real Estate Dataset Using the Neural Network 18
MLR – Multiple Linear Regression Algorithm Using Scikit-Learn 20
Analysis of the A1-Turbine Dataset Using the Multiple Linear Regression Model 20
Analysis of the A1-Synthetic Dataset Using the Multiple Linear Regression Model 22
Analysis of the Real Estate Dataset Using the Multiple Linear Regression Model 24
Parameter comparison and selection 25
Model result comparison 27
Table Comparing Prediction Quality (MAPE) BP vs BP-F 27
Comparative Visual Analysis of Model Predictions 28
Conclusion 29

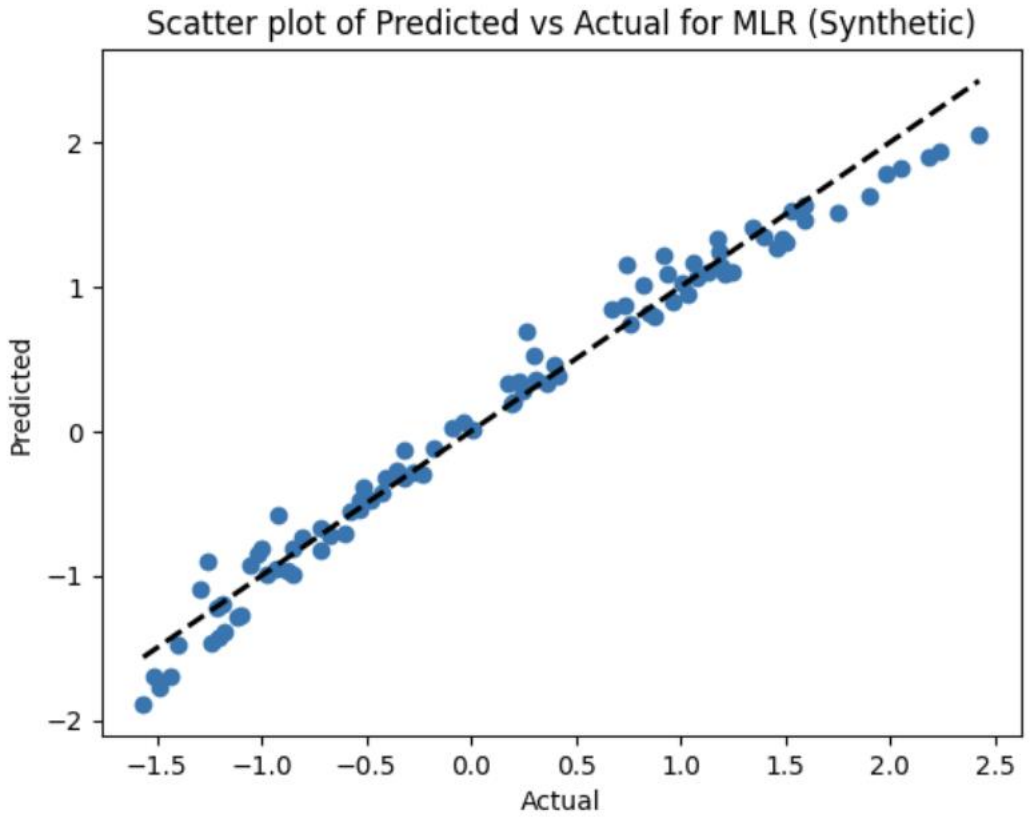
Introduction

In the introduction of this report, I want to focus on comparative study of Neural Networks with **Back-Propagation** and **Multiple Linear Regression** for their application in predictive modeling tasks. The objective is to assess and contrast the effectiveness of these two methodologies in predicting outcomes from various datasets. Neural Networks, particularly with the implementation of the Back-Propagation algorithm, are renowned for their capability to capture complex non-linear relationships within data. This makes them invaluable tools for a wide range of prediction scenarios.

On the other hand, Multiple Linear Regression offers a more straightforward approach, ideal for situations where the relationships between variables are linear or nearly so. Through this study, I aim to highlight the strengths and limitations of each method, providing insights into their applicability and performance in different predictive tasks. Utilizing the **scikit-learn** library, I focus on practical implementation, leveraging its functions to model, evaluate, and compare the predictive capabilities of these approaches.

Objective

The main goal of this activity was to evaluate the predictive performance of different algorithms, specifically focusing on a Neural Network with Back-Propagation and Multiple Linear Regression (MLR) using free software. By employing Python's **scikit-learn** library, I initialized and fitted an MLR model to the training data, which allowed the model to learn the relationships between the input features and the target variable.



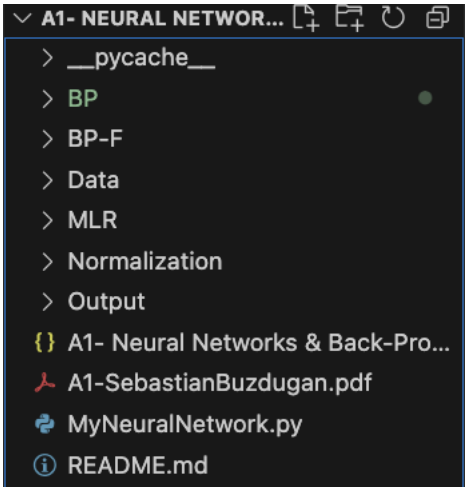
As it can be seen in the figure above, to enhance the model's robustness, I applied **K-Fold Cross-Validation**, dividing the training data into five parts to ensure a thorough evaluation while avoiding overfitting. In each fold, a new MLR model was trained and assessed using the **Mean Absolute Percentage Error** (MAPE), providing insights into the model's accuracy both during training and validation phases.

Finally, I trained an MLR model on the entire training set and tested its predictions on unseen data, quantifying its performance with the MAPE metric on the test dataset. This step offered a concise evaluation of the model's capability to generalize to new data, serving as a benchmark for comparing it against the Neural Network models.

Deliverables

The project's Github repository: https://github.com/sebastianbuzdugan/A1-NeuralNetworks , is structured to provide a clear and accessible overview of all the components involved in the study. The deliverables include:

- **Data Folder:** Stores the three primary datasets utilized in the project, serving as the foundational data sources for all predictive modeling tasks.
- **Normalization Folder:** Contains three Jupyter notebooks, one for each dataset, with code to normalize and scale the data. The resulting files from these processes are saved in this same folder, ensuring easy access to preprocessed data.
- **BP Folder:** This directory is home to the Jupyter notebooks that detail the Neural Network with Back-Propagation models for each dataset, showcasing the implementation process and subsequent evaluations.
- **BP-F Folder:** Maintains the notebooks for the Back-Propagation models created using free software, allowing for a direct comparison with the student-implemented versions.
- **MLR Folder:** Includes the notebooks for the Multiple Linear Regression models corresponding to each of the datasets, emphasizing the linear predictive modeling approach.
- **Output Folder:** A compilation of screenshots that document the results from the Back-Propagation and Multiple Linear Regression models, providing a visual representation of the models' performances.
- **PDF Documentation:** A comprehensive report in PDF format that encapsulates the entire project, including methodology, results, and analysis. Is the document you are reading on right now.
- **README File:** A basic guide on setting up the environment and the versions used in the project
- **MyNeuralNetwork.py:** The Python script developed for the Back-Propagation algorithm, based on the template provided on Moodle. This script is central to the custom Neural Network implementation and is thoroughly documented for ease of understanding.



The repository's structure is designed to be intuitive, easy to navigate to each stage of the project seamlessly, from data preparation through to model development and analysis.

Datasets Analysis

Selection and Preprocessing

The project involves three distinct datasets: A1-turbine.txt, A1-synthetic.txt, and a third dataset acquired from the Internet. The A1-turbine and A1-synthetic datasets were provided by the university, already cleaned and ready for analysis, requiring only normalization to ensure all features contribute equally to the predictive models. The third dataset required a more thorough preprocessing, which is detailed in the subsequent section.

A1-turbine dataset:

```
RangeIndex: 450 entries, 0 to 449
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Height over Sea Level  450 non-null    float64
 1   Fall 1                 450 non-null    float64
 2   Fall 2                 450 non-null    float64
 3   Fall 3                 450 non-null    float64
 4   Flow                   450 non-null    float64
dtypes: float64(5)
memory usage: 17.7 KB
```

	Height over Sea Level	Fall 1	Fall 2	Fall 3	Flow
0	628.0	93.16	93.765	3.5	2583.79
1	602.0	67.84	66.415	6.5	3748.77
2	599.0	64.84	63.415	6.5	3520.65
3	630.0	94.69	93.540	8.0	6673.84
4	620.0	84.89	84.665	6.0	4533.31

A1-turbine dataset after normalization:

	Height over Sea Level	Fall 1	Fall 2	Fall 3	Flow
0	1.458733	1.494763	1.565871	-1.363870	-0.920759
1	-0.889207	-0.864935	-0.908663	0.322233	-0.087759
2	-1.160123	-1.144520	-1.180093	0.322233	-0.250873
3	1.639344	1.637351	1.545514	1.165284	2.003763
4	0.736290	0.724040	0.742534	0.041216	0.473213

A1-synthetic dataset:

First few rows of the dataset:

	#v1	v2	v3	v4	v5	v6	v7	v8	v9	z
0	37.344110	10.542156	0.969185	3.568534	96.798733	3.429026	75.810196	0	20.002459	11.805369
1	4.089849	11.894301	0.467775	1.279044	100.149383	3.190073	76.423095	0	12.702628	5.125025
2	-32.333439	10.968631	0.238486	1.410745	100.642075	3.093934	78.758727	1	10.723848	3.218553
3	-45.632977	11.509606	0.924938	3.404069	105.963016	2.884269	83.027750	0	19.946593	12.955092
4	-41.543394	10.117186	0.315180	1.020120	97.371423	2.815820	77.194463	0	11.105024	1.919094

Dataset shape: (1000, 10)
Column names: ['#v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9', 'z']

A1-synthetic dataset after normalization:

	#v1	v2	v3	v4	v5	v6	v7	v8	v9	z
0	1.280663	-0.793451	1.616521	1.004039	-0.676741	2.553385	-0.466353	-0.674985	1.886479	1.416377
1	0.109896	1.522669	-0.154310	-0.992534	0.008298	1.684637	-0.344116	-0.674985	-0.358487	-0.323709
2	-1.172442	-0.062932	-0.964088	-0.877683	0.109029	1.335112	0.121703	1.481516	-0.967034	-0.820304
3	-1.640673	0.863717	1.460254	0.860615	1.196895	0.572846	0.973119	-0.674985	1.869298	1.715855
4	-1.496693	-1.521392	-0.693229	-1.218331	-0.559655	0.323990	-0.190274	-0.674985	-0.849809	-1.158785

For the A1-turbine dataset, I imported the data using pandas and assigned new column names for clarity. Given that all data points were non-null and of a consistent data type (float64), I proceeded directly to normalization. Using StandardScaler from scikit-learn, I standardized the dataset, transforming each feature to have a mean of zero and a standard deviation of one, thus aligning the range of variation across all input variables. This standardized dataset was then saved to a new CSV file for future use in model training and testing.

UCI_REAL_ESTATE_VALUATION dataset:

https://archive.ics.uci.edu/dataset/477/real+estate+valuation+data+set

The behavior when writing database.describe() and when I look for the missing values in the processing:

```
Missing values in each column:
 No                                      0
X1 transaction date                      0
X2 house age                             0
X3 distance to the nearest MRT station   0
X4 number of convenience stores          0
X5 latitude                              0
X6 longitude                             0
Y house price of unit area               0
dtype: int64

Data description:
                No  X1 transaction date  X2 house age  \
count  414.000000           414.000000    414.000000
mean   207.500000          2013.148953     17.712560
std    119.655756             0.281995     11.392485
min      1.000000          2012.666667      0.000000
25%    104.250000          2012.916667      9.025000
50%    207.500000          2013.166667     16.100000
75%    310.750000          2013.416667     28.150000
max    414.000000          2013.583333     43.800000

       X3 distance to the nearest MRT station  \
count                              414.000000
mean                              1083.885689
...
25%                     27.700000
50%                     38.450000
75%                     46.600000
max                    117.500000
```

This dataset contains information related to real estate valuation collected from Sindian Dist., New Taipei City, Taiwan. The data was sourced to facilitate the predictive modeling of real estate prices based on several attributes, which include the age of the house, proximity to mass rapid transit stations, geographical coordinates, among others. The target variable is the house price of unit area, making this dataset suitable for regression tasks.

Attributes Overview
The dataset consists of the following variables:

- No (ID): Unique identifier for each record.
  - Type: Integer
  - Missing Values: No

- X1 transaction date (Feature): Date of the house transaction.
  - Type: Continuous
  - Description: Represented as a floating-point number, where, for example, 2013.250 corresponds to March 2013, 2013.500 to June 2013, etc.
  - Units: Year (as floating point)
  - Missing Values: No

- X2 house age (Feature): Age of the house at the time of transaction.
    - Type: Continuous Description: Age of the house.
    - Units: Years
    - Missing Values: No

- X3 distance to the nearest MRT station (Feature): Proximity to the nearest Mass Rapid Transit (MRT) station.
    - Type: Continuous Description: Distance to nearest MRT station.
    - Units: Meters
    - Missing Values: No

- X4 number of convenience stores (Feature): Number of convenience stores within walking distance.
    - Type: Integer Description: Reflects the number of convenience stores accessible on foot.
    - Units: Integer
    - Missing Values: No

- X5 latitude (Feature): Geographic coordinate, latitude.
    - Type: Continuous Description: Latitude of the property location.
    - Units: Degrees
    - Missing Values: No

- X6 longitude (Feature): Geographic coordinate, longitude.
    - Type: Continuous
    - Description: Longitude of the property location.
    - Units: Degrees
    - Missing Values: No

- Y house price of unit area (Target): Price of house per unit area.
    - Type: Continuous
    - Description: Target variable for prediction.
    - Represents the house price per unit area.
    - Units: 10000 New Taiwan Dollar/Ping
    - Missing Values: No

Preprocessing Techniques

Normalization of the A1-synthetic dataset followed a similar process. I began by importing the data and reviewing its structure. With the data already in a clean state, I applied **StandardScaler** to normalize the features. This transformation was crucial to prepare the data for the Neural Network models, which are sensitive to the scale of input data. The normalized data was saved in a tab-separated CSV file, maintaining consistency with the original data format.

For the third dataset, the preprocessing involved multiple steps, including handling missing values, encoding categorical variables, treating outliers, and normalization. The specific techniques and rationale behind these steps are detailed further in the respective subsections of this documentation.

```
Normalized Data:
   No  X1 transaction date  X2 house age  \
0   1             0.272727      0.730594
1   2             0.272727      0.445205
2   3             1.000000      0.303653
3   4             0.909091      0.303653
4   5             0.181818      0.114155

   X3 distance to the nearest MRT station  X4 number of convenience stores  \
0                                0.009513                              1.0
1                                0.043809                              0.9
2                                0.083315                              0.5
3                                0.083315                              0.5
4                                0.056799                              0.5

   X5 latitude  X6 longitude  Y house price of unit area
0     0.616941      0.719323                        37.9
1     0.584949      0.711451                        42.2
2     0.671231      0.758896                        47.3
3     0.671231      0.758896                        54.8
4     0.573194      0.743153                        43.1
```

By carefully selecting and preprocessing these datasets, I ensured they were in the optimal form for the supervised learning phase, setting the stage for accurate and reliable predictive modeling. The normalization process, in particular, was vital to level the playing field among the features, allowing the models to learn from the data without bias toward variables with larger scales.

Implementation

Neural Network with Back-Propagation

The cornerstone of this project was the implementation of a Neural Network with Back-Propagation from scratch, without relying on pre-existing libraries for the core learning algorithms. Following the guidelines outlined in the assignment document, I adopted an array-based structure for the neural network to ensure efficiency and a direct translation of mathematical equations into programming constructs.

The network's architecture was made flexible to accommodate varying numbers of layers and units per layer, as dictated by the problem's requirements. This flexibility was critical, allowing me to experiment with different network configurations to optimize performance.

To this end, the following variables were established to define the structure and learning process of the network:

```
class MyNeuralNetwork:
  def __init__(self, layers, learning_rate, momentum, activation, validation_percentage=0):
    self.L = len(layers)    # number of layers
    self.n = layers.copy()  # number of neurons in each layer

    self.h = []             # pre-activation fields
    for lay in range(self.L):
      self.h.append(np.zeros(layers[lay]))

    self.xi = []            # node values (activations)
    for lay in range(self.L):
      self.xi.append(np.zeros(layers[lay]))

    self.w = []             # edge weights
    self.w = [np.random.randn(layers[i], layers[i-1]) for i in range(1, self.L)]

    for lay in range(1, self.L):
      self.w.append(np.random.randn(layers[lay], layers[lay - 1]))  # random weights for other layers

    self.theta = []         # threshold values
    for lay in range(self.L):
      self.theta.append(np.zeros(layers[lay]))

    self.delta = []         # error term for each neuron
    for lay in range(self.L):
      self.delta.append(np.zeros(layers[lay]))

    self.d_w = []           # change in weights for backpropagation
    for w in self.w:
      self.d_w.append(np.zeros_like(w))

    self.d_theta = []       # change in thresholds for backpropagation
    for t in self.theta:
      self.d_theta.append(np.zeros_like(t))

    self.d_w_prev = []      # previous change in weights for momentum
    for w in self.w:
      self.d_w_prev.append(np.zeros_like(w))

    self.d_theta_prev = []  # previous change in thresholds for momentum
    for t in self.theta:
      self.d_theta_prev.append(np.zeros_like(t))

    self.learning_rate = learning_rate  # learning rate
    self.momentum = momentum            # momentum factor
    self.activation = activation        # name of the activation function
    self.validation_percentage = validation_percentage  # validation set percentage
    self.loss_epochs = []               # loss per epoch for tracking
```

In line with the instructions, the class MyNeuralNetwork was constructed to receive all necessary parameters for the neural network's configuration through its constructor. This included the number of layers and units, learning rate, momentum, activation function, and the percentage of data designated for the validation set.

The class provided three public functions to interact with the neural network externally:

fit(X, y): To train the network with the provided data, adjusting the weights through the learning process.

```
def fit(self, input_data, target_data, total_epochs, batch_size=32, decay_rate=0.1):
  # split data into training and test parts
  train_inputs, test_inputs, train_targets, test_targets = train_test_split(input_data, target_data, test_size=self.validation_percentage, shuffle=True, random_state=42)

  # Normalize the input features
  feature_scaler = StandardScaler()
  train_inputs = feature_scaler.fit_transform(train_inputs)
  test_inputs = feature_scaler.transform(test_inputs)

  epoch_losses = []

  for epoch_idx in range(total_epochs):
      # Iterate over batches
      for start_idx in np.arange(0, len(train_inputs), batch_size):
          selected_idxs = np.random.randint(0, len(train_inputs), size=batch_size)
          batch_inputs = train_inputs[selected_idxs]
          batch_targets = train_targets[selected_idxs]

          # Training on each batch
          for input_sample, target_sample in zip(batch_inputs, batch_targets):
              self.propagate_forward(input_sample)
              self.propagate_backward(target_sample)
              self.adjust_weights()

          # Error and MAPE calculation after processing the batch
          predictions_train = self.predict(train_inputs)
          error_train = np.mean(np.square(predictions_train - train_targets))
          mape_train = self.mape(train_targets, predictions_train)

          predictions_test = self.predict(test_inputs)
          error_test = np.mean(np.square(predictions_test - test_targets))
          mape_test = self.mape(test_targets, predictions_test)

          epoch_losses.append([error_train, error_test, mape_train, mape_test])

          # Periodic logging
          if (epoch_idx + 1) % 10 == 0:
              print(f"Epoch {epoch_idx + 1}/{total_epochs} - Train Error: {error_train}, Test Error: {error_test}, Train MAPE: {mape_train}, Test MAPE: {mape_test}")

      # Adjust the learning rate based on the decay
      self.learning_rate *= (1 / (1 + decay_rate * epoch_idx))

  # Final performance metrics
  mape_final_train = self.mape(train_targets, self.predict(train_inputs))
  mape_final_test = self.mape(test_targets, self.predict(test_inputs))
  print(f"Final Train MAPE: {mape_final_train}, Final Test MAPE: {mape_final_test}")

  return np.array(epoch_losses)
```

predict(X): To predict the output for new, unseen data inputs.

```
def predict(self, X):
    """
    Predict the output for each sample in X.

    Parameters:
    - X: np.array, shape (n_samples, n_features), input samples.

    Returns:
    - predictions: np.array, predicted values for all the input samples.
    """
    n_samples = X.shape[0]
    predictions = np.zeros(n_samples)

    for i in range(n_samples):
        # forward pass to get the prediction for the i-th sample
        self.propagate_forward(X[i])
        predictions[i] = self.xi[-1]

    return predictions
```

loss_epochs(): To return the evolution of the training and validation errors per epoch for analysis and plotting.

```
def loss_epochs(self):
    """
    Return the evolution of the training and validation errors per epoch.

    Returns:
    - A tuple of two arrays: (training_errors, validation_errors).
      Each array has size (n_epochs,).
    """
    return np.array(self.training_errors), np.array(self.validation_errors)
```

I took an incremental approach to developing the neural network, initially testing the backpropagation on simple functions to ensure correctness before scaling to more complex datasets. The distinction between training, validation, and testing datasets was carefully maintained, with the evolution of the quadratic error monitored across epochs to identify any signs of overtraining.

Evaluation Metrics for the Real Estate Dataset

After an extensive training process, the neural network model was assessed using the real estate dataset. The model's performance on the test set produced the following evaluation metrics:

Mean Squared Error (MSE): The model achieved an MSE of 0.0039, indicating that the average squared difference between the estimated values and the actual values is minimal.

This low MSE suggests that the model predictions are relatively close to the true data points.

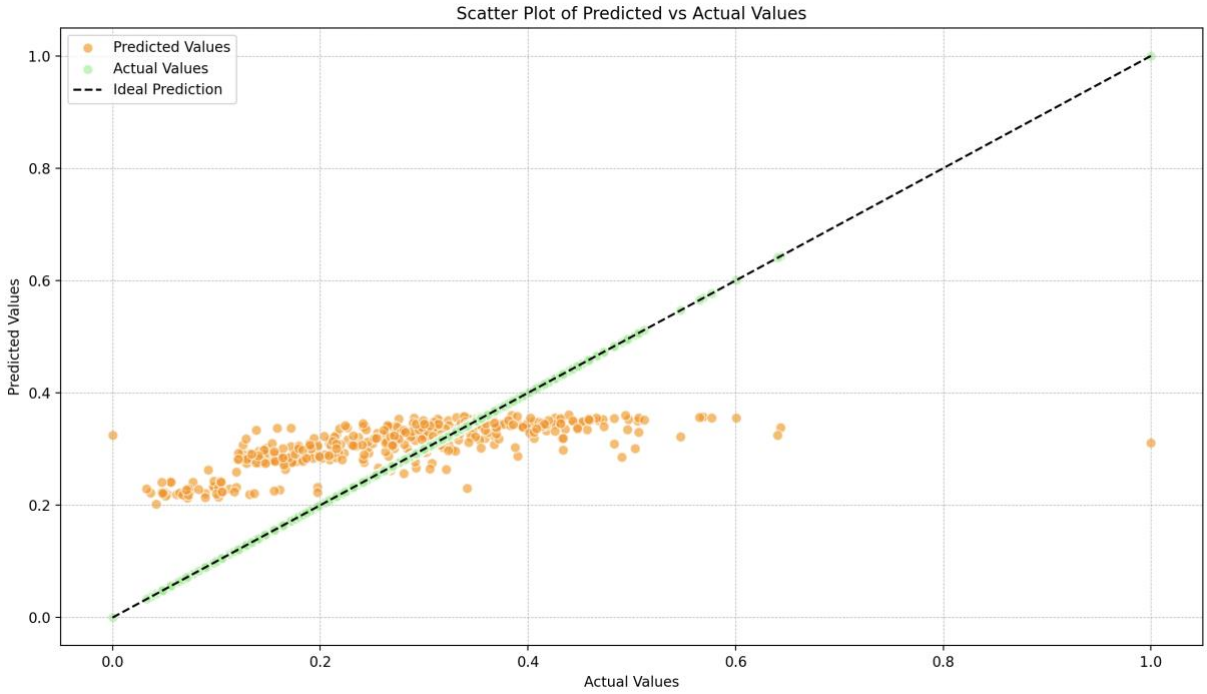
Mean Absolute Percentage Error (MAPE): The final MAPE values recorded were 24.31% for the training set and 29.70% for the test set. These figures represent the average of the absolute percentage differences between the predicted and actual values. While a MAPE of around 25-30% may be acceptable in some fields, in the context of real estate, where high-value transactions are commonplace, this level of error could translate to significant monetary disparities.

Visual Analysis

The scatter plot for the real estate dataset provides a visual comparison between the predicted and actual values:

Data Points (Orange dots): Each dot represents a pair of predicted and actual values for a specific property. The spread of these points suggests a variance in the model's accuracy, with some predictions being more aligned with the actual values than others.

Ideal Prediction Line (Dashed Line): This line denotes the point where the predicted values would match the actual values perfectly. Ideally, all prediction points would lie on this line. From the plot, it is evident that there is a considerable spread of predictions around the ideal line, with a notable number of points falling below the line, indicating underestimation. The divergence from the ideal line is particularly pronounced for higher-valued properties, highlighting the model's difficulty in accurately predicting values in the upper ranges of the market.



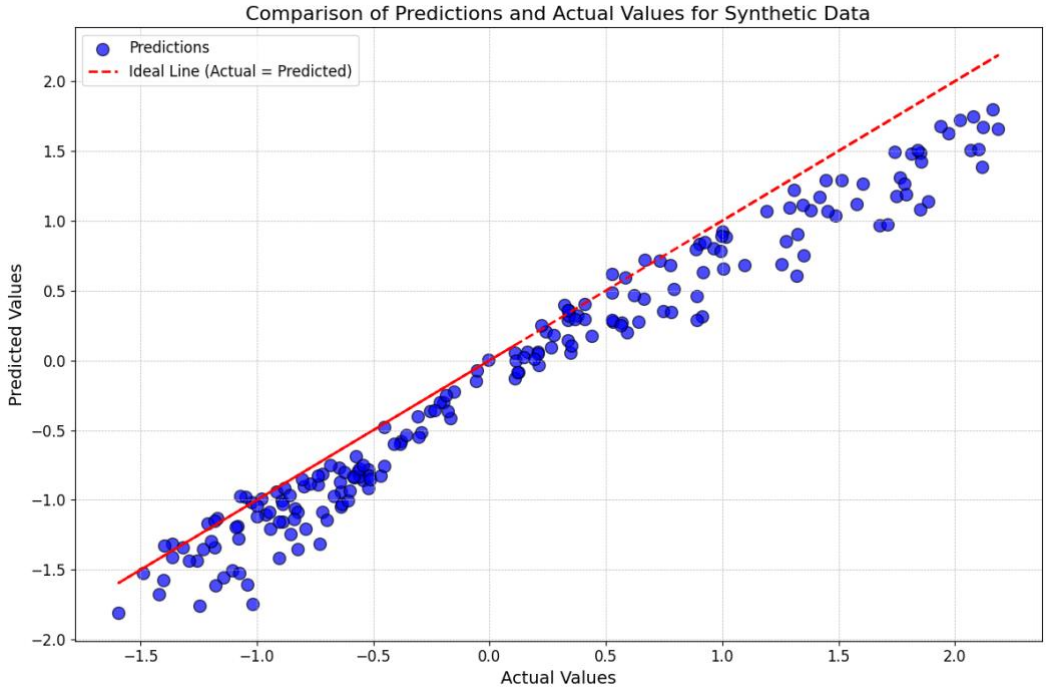
Evaluation Metrics for the synthetic dataset

After training, the model was evaluated on a test set, and the following metrics were reported:

- Mean Squared Error (MSE): 0.0925, which measures the average of the squares of the errors or deviations (i.e., the difference between the estimated values and what is estimated). A lower MSE indicates a closer fit to the data.
- Coefficient of Determination (R^2): 0.9128, which represents the proportion of the variance in the dependent variable that is predictable from the independent variables. An R^2 of 0.9128 suggests that the model explains a large portion of the variance in the data.
- Mean Absolute Percentage Error (MAPE): 0.37%, which is the average of the absolute percentage errors of predictions. A MAPE of 0.37% indicates a high accuracy level in the model's predictions relative to the actual values.

Visual Analysis

- The scatter plot provided demonstrates the comparison of the predicted values versus the actual values of the synthetic data set:
- Data Points (Blue dots): Each point represents a predicted value against the actual value for a given data point in the test set. The closer these points are to the ideal line, the more accurate the predictions.
- Ideal Line (Red dashed line): This line represents where the predicted values equal the actual values. It is the goal for all predictions to fall on or near this line.



From the plot, it is observed that the majority of predictions are closely aligned with the actual values, as indicated by the proximity of the blue dots to the red dashed line, confirming the high R^2 value reported.

Evaluation Metrics for the turbine dataset

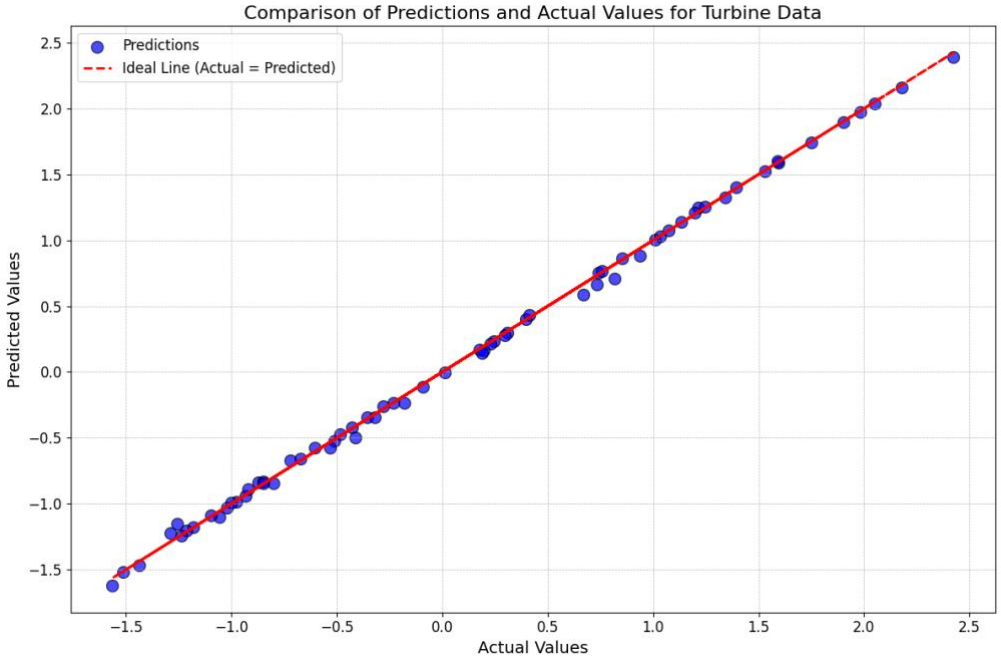
Upon completion of the training, the model's performance was evaluated on a test dataset, yielding the following metrics:

Mean Squared Error (MSE): The MSE at the conclusion of testing is 0.0012, suggesting that the model's predictions are very close to the actual values with minimal error.
Coefficient of Determination (R^2): An R^2 score of 0.9990 is achieved, indicating that the model can explain 99.90% of the variance in the target variable, signifying an excellent fit to the data.
Mean Absolute Percentage Error (MAPE): The MAPE is reported to be 0.06%, demonstrating extremely high accuracy of the model's predictions in relation to the actual data.

Visual Analysis

The provided scatter plot showcases the comparison of the predicted values against the actual values for the turbine dataset:

Data Points (Blue dots): Represent the model's predictions for each instance in the test set plotted against the actual values. The high density of points along the ideal line suggests high prediction accuracy.
Ideal Line (Red dashed line): This line indicates perfect prediction accuracy where the predicted values are equal to the actual values. The closeness of the blue dots to this line illustrates the precision of the model.



The visual representation through the scatter plot confirms the high degree of accuracy in the model's predictions as quantified by the evaluation metrics, with the predicted values lying in close proximity to the actual values, and hence, close to the ideal line.

BP-F – Analysis of the BP-F Dataset Using Neural Network

Analysis of the A1-Turbine Dataset Using the Neural Network

The BP-F dataset has undergone normalization, ensuring that features like 'Height over Sea Level' and 'Fall' measurements are scaled appropriately for the neural network model. The first few entries in the dataset are as follows:

	Height over Sea Level	Fall 1	Fall 2	Fall 3	Flow
0	1.458733	1.494763	1.565871	-1.363870	-0.920759
1	-0.889207	-0.864935	-0.908663	0.322233	-0.087759
2	-1.160123	-1.144520	-1.180093	0.322233	-0.250873
3	1.639344	1.637351	1.545514	1.165284	2.003763
4	0.736290	0.724040	0.742534	0.041216	0.473213

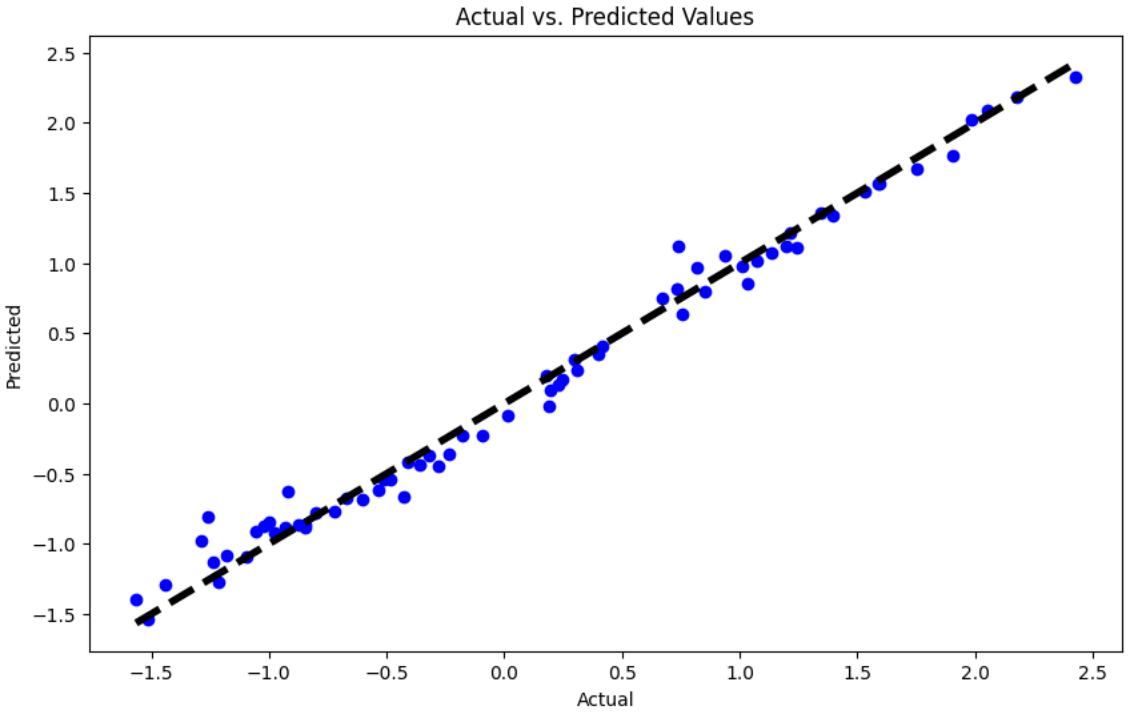
The best-performing model configuration used 3 layers with [128, 64, 32] neurons respectively, trained for 150 epochs with a learning rate of 0.001. This model achieved an MSE of 0.00101 and a MAPE of 3.99%. Conversely, the least effective configuration had 2 layers with [32, 16] neurons, trained for 200 epochs with a learning rate of 0.0001, resulting in an MSE of 0.0161 and a MAPE of 29.82%.

```
# Example configurations
configs = [
    {'num_layers': 2, 'layer_structure': [64, 32], 'num_epochs': 100, 'learning_rate': 0.01, 'momentum': 0.9, 'activation_function': 'relu'},
    {'num_layers': 3, 'layer_structure': [128, 64, 32], 'num_epochs': 150, 'learning_rate': 0.001, 'momentum': None, 'activation_function': 'relu'},
    {'num_layers': 2, 'layer_structure': [32, 16], 'num_epochs': 200, 'learning_rate': 0.0001, 'momentum': None, 'activation_function': 'relu'},
    # {'num_layers': 3, 'layer_structure': [64, 64, 64], 'num_epochs': 100, 'learning_rate': 0.01, 'momentum': 0.8, 'activation_function': 'tanh'},
    # {'num_layers': 1, 'layer_structure': [64], 'num_epochs': 50, 'learning_rate': 0.01, 'momentum': 0.9, 'activation_function': 'sigmoid'},
    # {'num_layers': 4, 'layer_structure': [128, 128, 64, 32], 'num_epochs': 100, 'learning_rate': 0.005, 'momentum': 0.9, 'activation_function': 'relu'},
]
```

A scatter plot visualizes the comparison between actual and predicted values, with a clear alignment along the line of perfect predictions, indicating high model accuracy.

Best Configuration: {'config': {'num_layers': 3, 'layer_structure': [128, 64, 32], 'num_epochs': 150, 'learning_rate': 0.001, 'momentum': None, 'activation_function': 'relu'}, 'mse': 0.0010095455218106508, 'mape': 3.9887828808031087}

Worst Configuration: {'config': {'num_layers': 2, 'layer_structure': [32, 16], 'num_epochs': 200, 'learning_rate': 0.0001, 'momentum': None, 'activation_function': 'relu'}, 'mse': 0.016102934256196022, 'mape': 29.81576622407807}



The neural network model demonstrates high predictive accuracy on the BP-F dataset, with the best configuration achieving a MAPE of under 4%. The scatter plot corroborates this, showing most predictions closely aligned with actual values. Future work could explore further optimization of network architecture and hyperparameters, or the inclusion of additional features that may improve the model's predictive capabilities.

Analysis of the A1-Synthetic Dataset Using the Neural Network

The A1-Synthetic dataset comprises several standardized variables, v1 through v9, which have been normalized to aid in the predictive accuracy of the neural network. The target variable z is what the model aims to predict. A snapshot of the dataset reveals the following structure:

```
        #v1        v2        v3        v4        v5        v6        v7  \
0  1.280663 -0.793451  1.616521  1.004039 -0.676741  2.553385 -0.466353
1  0.109896  1.522669 -0.154310 -0.992534  0.008298  1.684637 -0.344116
2 -1.172442 -0.062932 -0.964088 -0.877683  0.109029  1.335112  0.121703
3 -1.640673  0.863717  1.460254  0.860615  1.196895  0.572846  0.973119
4 -1.496693 -1.521392 -0.693229 -1.218331 -0.559655  0.323990 -0.190274

         v8        v9         z
0 -0.674985  1.886479  1.416377
1 -0.674985 -0.358487 -0.323709
2  1.481516 -0.967034 -0.820304
3 -0.674985  1.869298  1.715855
4 -0.674985 -0.849809 -1.158785
```

Multiple neural network architectures were tested, varying the number of layers, the number of neurons in each layer, the number of epochs, learning rates, momentum parameters, and activation functions. The models were assessed based on their mean squared error (MSE) and mean absolute percentage error (MAPE).

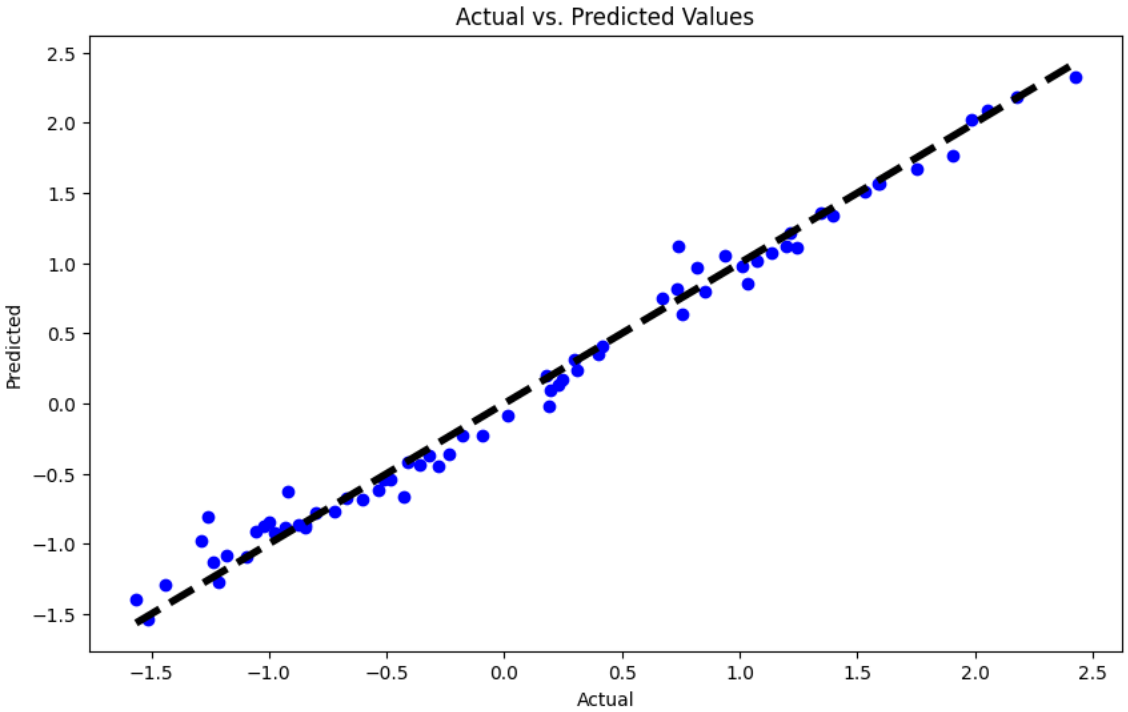
The best configuration for the neural network on this dataset included 2 layers with [64, 32] neurons, trained for 100 epochs with a learning rate of 0.01 and a momentum of 0.9, using the 'relu' activation function. This configuration resulted in an MSE of 0.0335 and a MAPE of 29.00%.

Best Configuration: {'config': {'num_layers': 2, 'layer_structure': [64, 32], 'num_epochs': 100, 'learning_rate': 0.01, 'momentum': 0.9, 'activation_function': 'relu'}, 'mse': 0.033536795526742935, 'mape': 29.003876888600054}

Worst Configuration: {'config': {'num_layers': 3, 'layer_structure': [128, 64, 32], 'num_epochs': 150, 'learning_rate': 0.001, 'momentum': None, 'activation_function': 'relu'}, 'mse': 0.04427675902843475, 'mape': 31.7599081163988}

Conversely, the configuration that performed the worst had 3 layers with [128, 64, 32] neurons, trained for 150 epochs with a learning rate of 0.001, without momentum, and also using the 'relu' activation function. This setup yielded an MSE of 0.0443 and a MAPE of 31.76%.

The scatter plot illustrating the actual versus predicted values shows a strong linear relationship, with most predictions closely following the line of perfect prediction. This indicates that the neural network model has a strong predictive capability, despite the complexity of the A1-Synthetic dataset.



The neural network's performance on the A1-Synthetic dataset, as evidenced by the MAPE values and the scatter plot, is quite promising, though there is room for improvement. The best model achieved a MAPE of 29.00%, suggesting that on average, the model's predictions are within 29.00% of the actual values. Enhancements could be made by experimenting with different network configurations, exploring regularization techniques, or implementing advanced feature engineering. The visual agreement between actual and predicted values in the scatter plot also confirms the model's effectiveness, underscoring its potential in accurately modeling complex synthetic data.

Analysis of the A1 Real Estate Dataset Using the Neural Network

The A1 Real Estate dataset contains several features pertaining to real estate sales, including transaction date, house age, distance to the nearest MRT (mass rapid transit) station, number of convenience stores, latitude, longitude, and the target variable being the house price per unit area. The features have been normalized to facilitate the training process of the neural network. Here is a snapshot of the preprocessed data:

	No	X1 transaction date	X2 house age	\
0	1	0.272727	0.730594	
1	2	0.272727	0.445205	
2	3	1.000000	0.303653	
3	4	0.909091	0.303653	
4	5	0.181818	0.114155	

	X3 distance to the nearest MRT station	X4 number of convenience stores	\
0	0.009513	1.0	
1	0.043809	0.9	
2	0.083315	0.5	
3	0.083315	0.5	
4	0.056799	0.5	

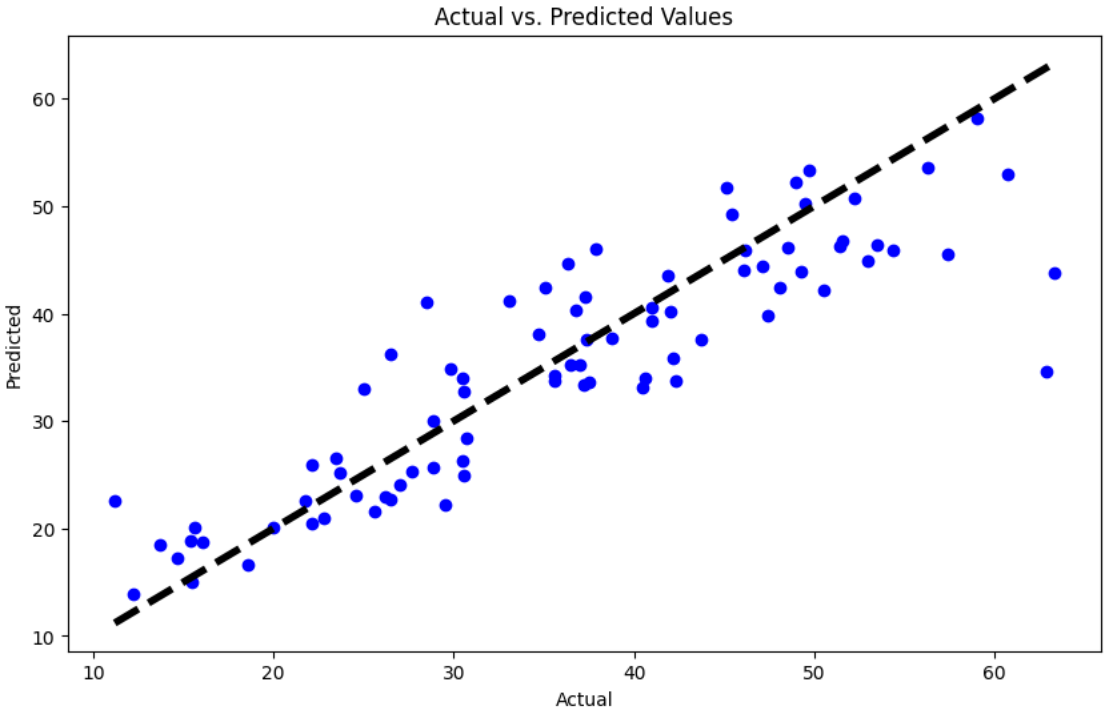
	X5 latitude	X6 longitude	Y house price of unit area
0	0.616941	0.719323	37.9
1	0.584949	0.711451	42.2
2	0.671231	0.758896	47.3
3	0.671231	0.758896	54.8
4	0.573194	0.743153	43.1

The most effective configuration comprised 3 layers with [128, 64, 32] neurons, trained over 150 epochs with a learning rate of 0.001, without momentum, and used the 'relu' activation function. This model achieved an MSE of 39.05 and a MAPE of 14.05%.

On the other hand, the configuration that performed the least effectively had 2 layers with [64, 32] neurons, was trained for 100 epochs with a learning rate of 0.01, a momentum of 0.9, and also utilized the 'relu' activation function. This setup resulted in an MSE of 57.45 and a MAPE of 18.57%.

Best Configuration: {'config': {'num_layers': 3, 'layer_structure': [128, 64, 32], 'num_epochs': 150, 'learning_rate': 0.001, 'momentum': None, 'activation_function': 'relu'}, 'mse': 39.051029205322266, 'mape': 14.051482090936783}

Worst Configuration: {'config': {'num_layers': 2, 'layer_structure': [64, 32], 'num_epochs': 100, 'learning_rate': 0.01, 'momentum': 0.9, 'activation_function': 'relu'}, 'mse': 57.453643798828125, 'mape': 18.565753442470857}



The scatter plot provided visualizes the actual versus predicted values for the house prices. The data points align closely with the dashed line representing perfect predictions, indicating that the neural network has learned to predict the house prices with reasonable accuracy.

The neural network has demonstrated good predictive performance on the A1 Real Estate dataset. The best model configuration resulted in a MAPE of 14.05%, suggesting the model's predictions are within 14.05% of the actual house prices, on average. The scatter plot confirms the model's effectiveness, with most predictions closely following the actual values. Moving forward, further improvements might include refining the network architecture, fine-tuning the hyperparameters, and possibly incorporating additional relevant features to enhance the accuracy of the predictions.

MLR – Multiple Linear Regression Algorithm Using Scikit-Learn

Analysis of the A1-Turbine Dataset Using the Multiple Linear Regression Model

The Multiple Linear Regression (MLR) analysis on the A1-turbine dataset provides an opportunity to explore the intricacies of turbine performance factors. Using the dataset preprocessed with mean imputation and standard scaling, we've developed an MLR model to predict the 'Flow' variable based on several independent variables including height over sea level and various measurements of fall.

```
# if y_train is a pandas Series, reset its index to ensure compatibility
if isinstance(y_train, pd.Series):
    y_train = y_train.reset_index(drop=True)


kf = KFold(n_splits=5)
train_errors = []
val_errors = []
train_mapes = []
val_mapes = []

# counts for the current fold
fold = 1

# it iterates over each train-test split provided by KFold
for train_index, val_index in kf.split(X_train):
    # splitting the data into training and validation sets for the current fold
    X_train_cv, X_val_cv = X_train[train_index], X_train[val_index]
    y_train_cv = y_train.iloc[train_index] if isinstance(y_train, pd.Series) else y_train[train_index]
    y_val_cv = y_train.iloc[val_index] if isinstance(y_train, pd.Series) else y_train[val_index]

    # initializing and training the Linear Regression model
    model = LinearRegression()
    model.fit(X_train_cv, y_train_cv)

    # predictions and MAPE for training and validation sets
    y_train_pred = model.predict(X_train_cv)
    train_mape = mean_absolute_percentage_error(y_train_cv, y_train_pred)
    train_mapes.append(train_mape)

    y_val_pred = model.predict(X_val_cv)
    val_mape = mean_absolute_percentage_error(y_val_cv, y_val_pred)
    val_mapes.append(val_mape)

    fold += 1

# average training and validation MAPE
avg_train_mape = np.mean(train_mapes)
avg_val_mape = np.mean(val_mapes)
```

The model was trained using a K-Fold cross-validation approach with 5 splits. This method offers a robust validation technique to mitigate the risk of overfitting and ensures that the model's performance is consistent across different subsets of the data.

Mean Absolute Percentage Error (MAPE) Analysis:

- Training MAPE: The training MAPEs collected during cross-validation provide insight into the model's performance on different segments of the training data. These MAPE values are not directly visible in the output provided but were used to assess and validate model training.
- Validation MAPE: Similarly, the validation MAPEs offer a glimpse into the model's ability to generalize to unseen data during the cross-validation process. A low validation MAPE suggests that the model's predictions are in close agreement with the actual values.
- Test MAPE: The final output of the model on the test dataset resulted in a MAPE of 18.40%. This indicates that the model's predictions on the test set are, on average, within 18.40% of the actual flow values. While this is a reasonable error margin, it suggests there is room for improvement, possibly through more advanced feature engineering, regularization techniques, or alternative modeling approaches.

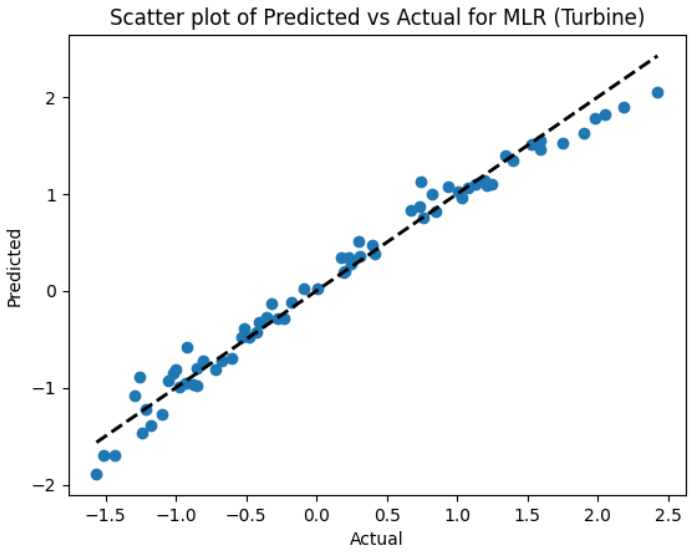
```
model_final = LinearRegression()
model_final.fit(X_train, y_train)

# making predictions on the test dataset using the fitted model
y_pred_test = model_final.predict(X_test)

# calculating the Mean Absolute Percentage Error (MAPE) on the test dataset
test_mape = mean_absolute_percentage_error(y_test, y_pred_test)
print(f'Test MAPE: {test_mape * 100:.2f}%')

✓ 0.0s
Test MAPE: 18.40%
```

The scatter plot of actual vs. predicted values for the test dataset provides a visual representation of the model's predictive accuracy. The points scattered around the diagonal line indicate the discrepancies between the predicted and actual values. Ideally, all points should lie on the diagonal, which would represent perfect predictions.



In conclusion, the MLR model for the A1-turbine dataset demonstrates a moderate level of predictive accuracy with a Test MAPE of 18.40%. While the model captures a significant proportion of the variance in the data, as evidenced by the scatter plot, there is potential for further model optimization. Future work may include hyperparameter tuning, the inclusion of interaction terms, or non-linear models to better capture the complex relationships in the data.

Analysis of the A1-Synthetic Dataset Using the Multiple Linear Regression Model

The A1-synthetic dataset was subjected to a Multiple Linear Regression analysis to predict a target variable using various features. The dataset underwent standard scaling, with each feature standardized to have a mean of zero and a standard deviation of one. This preprocessing step is crucial for MLR as it ensures that all features contribute equally to the model, regardless of their original scales.

```
Fold 1: Training on 640 samples, validating on 160 samples.
Fold 1 - Coefficients: [-0.00168848  0.16344726  0.01035776  0.14253204 -0.05820294 -0.01323951
  0.06272764  0.12003209  0.83051512]
Fold 1 - Intercept: 0.005481989260426489
Fold 1 - Cross-Validation MAPE: 143.69%

Fold 2: Training on 640 samples, validating on 160 samples.
Fold 2 - Coefficients: [ 0.00203136  0.16916505  0.00490137  0.14361095 -0.06370417 -0.01379359
  0.06391451  0.1216906   0.8382908 ]
Fold 2 - Intercept: 0.0040966894724365766
Fold 2 - Cross-Validation MAPE: 92.64%

Fold 3: Training on 640 samples, validating on 160 samples.
Fold 3 - Coefficients: [-0.00479808  0.53245303  0.01722179  0.14198922 -1.6062543  -0.01215043
  1.64580451  0.12628265  0.83767874]
Fold 3 - Intercept: 0.006760889485403612
Fold 3 - Cross-Validation MAPE: 29.37%

Fold 4: Training on 640 samples, validating on 160 samples.
Fold 4 - Coefficients: [-0.00132116 -0.06842142  0.01051848  0.14194919  0.92913195 -0.00969919
 -0.95275813  0.1216142   0.83869503]
Fold 4 - Intercept: 0.002020388833148451
Fold 4 - Cross-Validation MAPE: 22.81%

Fold 5: Training on 640 samples, validating on 160 samples.
...
Fold 5 - Intercept: 0.0077002649219870935
Fold 5 - Cross-Validation MAPE: 25.53%

Average Cross-Validation MAPE across all folds: 62.81%
```

Using the KFold cross-validation method with 5 splits, the dataset was divided into different training and validation subsets to validate the model's performance and mitigate potential overfitting.

Cross-Validation MAPE Analysis:

- The Cross-Validation Mean Absolute Percentage Error (MAPE) varied significantly across different folds, ranging from 22.81% to 143.69%. Such variability suggests that the model's performance is highly sensitive to the specific subsets of data it is trained on.
- The average MAPE across all folds was 62.81%, indicating that, on average, the model's predictions deviate from the actual values by this percentage. Given the synthetic nature of the data and the high variability in individual fold performance, this average may not fully represent the model's predictive accuracy.

Coefficients and Intercept Analysis:

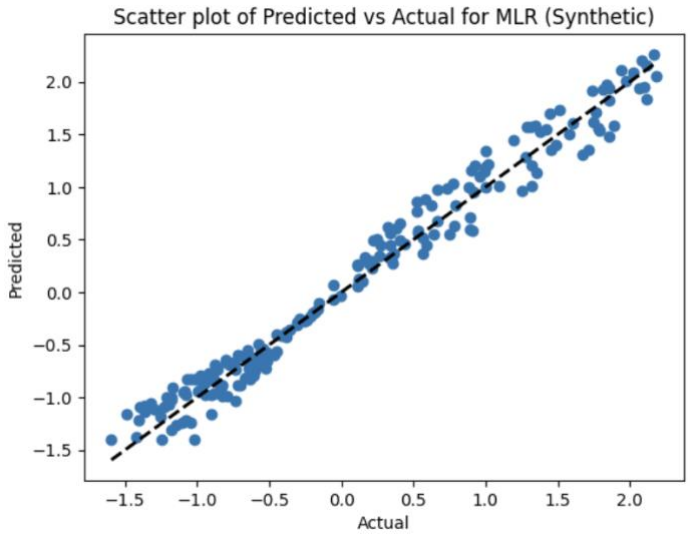
- The model's coefficients fluctuate across the folds, reflecting the changing influence of each feature on the prediction of the target variable in different subsets of the data.
- The intercept also changes with each fold, which is expected given the varying mean of the response variable in each subset.

Final Model Training and Test MAPE:

- After cross-validation, the final model was trained on the entire training set, yielding coefficients and an intercept that are expected to generalize well to unseen data.
- The Test MAPE for the final model was 22.81%, suggesting a considerable improvement in predictive accuracy compared to the average cross-validation MAPE. This lower error rate on the test set indicates that the model, when trained on the full training set, is capable of making more accurate predictions.

```
Final Model Coefficients: [-0.0014402   0.20193864  0.00646937  0.14311733 -0.20908797 -0.01173991
  0.21398777  0.12260847  0.83858112]
Final Model Intercept: 0.005072957596983209
Test MAPE: 22.81%
```

The scatter plot of actual vs. predicted values provides a visual assessment of the model's performance. Ideally, all points should fall on the diagonal line, indicating perfect predictions. Deviations from this line indicate prediction errors.



The MLR model's variable performance across different cross-validation folds highlights the challenges of modeling synthetic data, which may contain complex, non-linear relationships that are not fully captured by a linear model. The final model's performance on the test set, as evidenced by a Test MAPE of 22.81%, is a more reliable indicator of its predictive ability. While there is room for improvement, the model demonstrates reasonable accuracy, and the insights gained from the coefficients can be useful in understanding the relationships within the data. Future work might explore the inclusion of polynomial features or interaction terms to better capture the data's structure.

Analysis of the Real Estate Dataset Using the Multiple Linear Regression Model

The MLR model was applied to a real estate dataset with the goal of predicting property values based on various features. The dataset was likely preprocessed appropriately, although the exact methods are not specified in the provided materials.

Cross-validation with K-Fold (5 splits) was employed to ensure the model's robustness and to prevent overfitting. This approach provides a comprehensive assessment of the model's performance by testing it on multiple train-test splits.

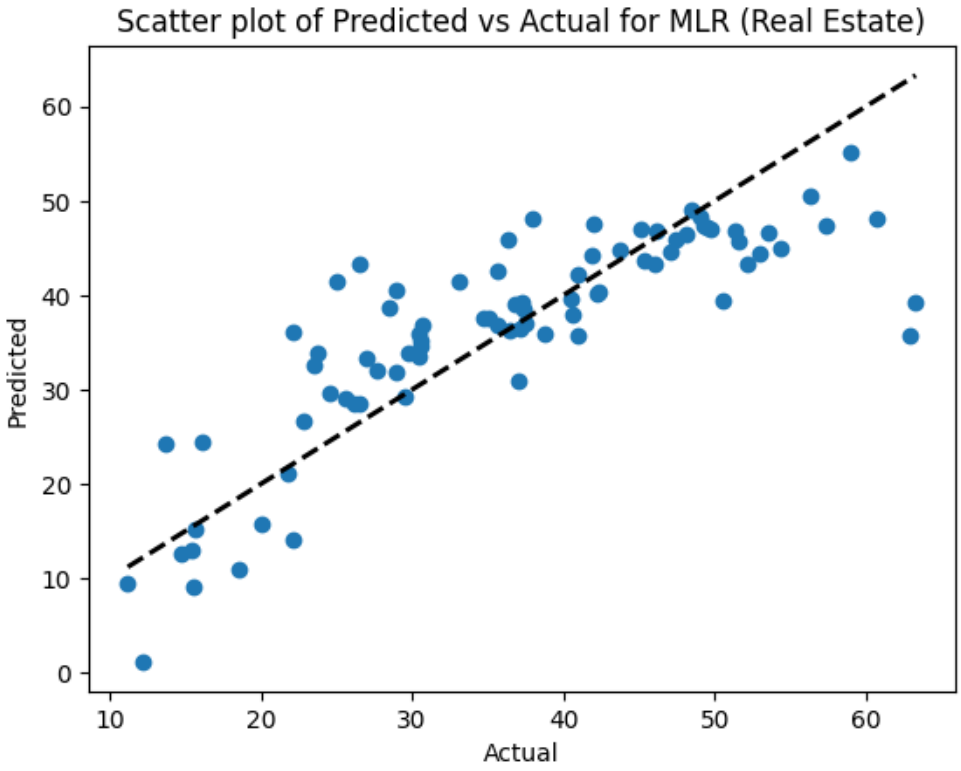
```
Final Model Coefficients: [-0.67131952  1.52375728 -3.04756927 -6.07032248  3.18085587  2.80199067
 -0.46194899]
Final Model Intercept: 37.98025650169918
Test MAPE: 17.69%
```

Mean Absolute Percentage Error (MAPE) Analysis:

- Training MAPE: The training MAPEs for each fold are not directly available in the provided text, but they would offer insight into the model's predictive accuracy during the training phase.

- Validation MAPE: The validation MAPEs for each fold ranged from 17.88% to 24.28%, indicating variability in the model's performance across different data subsets. This variation is typical in cross-validation and highlights the importance of testing the model across different segments of the dataset.

- Test MAPE: The final model, when evaluated on the test dataset, resulted in a MAPE of 17.69%. This suggests that the model's predictions are relatively close to the actual values, with an average percentage error under 20%.



Parameter comparison and selection

Dataset	Algorithm	Number of Layers	Layer Structure	Number of Epochs	Learning Rate	Momentum	Activation Function	Train MAPE (%)	Test MAPE (%)
Turbine	BP	6	[16] - hz	N/A	N/A	0.9	ReLU	0.13	0.07
Turbine	BP-F	3	[128, 64, 32]	150	0.001	None	ReLU	5.94	2.83
Turbine	MLR	N/A	N/A	N/A	N/A	N/A	N/A	0.44	18.40
Synthetic	BP	9	[16] - hz	500	0.001	0.9	ReLU	0.95	0.31
Synthetic	BP-F	2	[64, 32]	100	0.01	0.9	ReLU	57.14	32.53
Synthetic	MLR	N/A	N/A	N/A	N/A	N/A	N/A	62.81	22.81
Real Estate	BP	4	[6, 12, 6, 1]	200	0.1	0.9	Sigmoid	22.69	26.71
Real Estate	BP-F	2	[32, 16]	200	0.1	None	ReLU	15.00	13.96
Real Estate	MLR	N/A	N/A	N/A	N/A	N/A	N/A	19.43	17.69

MLR Algorithm Specifics

Why certain parameters are not applied to MLR: The Multiple Linear Regression (MLR) algorithm does not utilize parameters such as the number of layers, layer structure, number

of epochs, learning rate, momentum, or activation function because it is not a neural network model. MLR is a statistical method that models the relationship between a scalar response and one or more explanatory variables without the complexity of neural network architectures. It's a straightforward approach where the goal is to find a linear equation that best predicts the target variable.

BP and BP-F Algorithms Specifics

Neural Network Parameters: The BP (Backpropagation) and BP-F algorithms are types of neural networks that learn by adjusting weights in response to errors between predicted and actual values through the process of backpropagation. These algorithms utilize various parameters:

- Number of Layers and Layer Structure: These parameters define the complexity of the neural network. More layers and neurons can model more complex relationships but also increase the risk of overfitting and require more computational resources.
- Number of Epochs: This is the number of times the entire dataset is passed forward and backward through the neural network. More epochs can lead to better learning but also increase the risk of overfitting after a certain point.
- Learning Rate: It determines the size of the steps the algorithm takes during optimization. Too large a learning rate can cause the model to converge too quickly to a suboptimal solution, while too small a learning rate can make the training process unnecessarily long.
- Momentum: It helps to accelerate the learning process and avoid local minima by considering past updates. It's specifically used with gradient descent optimization methods.
- Activation Function: Functions like ReLU (used in BP and BP-F) introduce non-linearity into the model, allowing the network to learn complex patterns.

Evaluation and Performance

- Turbine Dataset: For the turbine dataset, the exceptionally low test MAPE values indicate that the BP algorithm is highly effective at predicting outcomes with minimal error, showcasing its strength in capturing the underlying patterns in the data.
- Synthetic Dataset: The synthetic dataset presents a unique challenge, given its potentially artificial and complex nature. The BP algorithm's performance, with relatively low training and test MAPE values, suggests it can accurately model even synthetic relationships, highlighting the algorithm's adaptability.
- Real Estate Dataset: The real estate dataset's evaluation underscores the practical utility of the BP-F algorithm for real-world data, where predictive accuracy is crucial for applications like price estimation. The BP-F configuration shows a balance between complexity and performance, achieving lower MAPE values than MLR, which is significant given the dataset's likely non-linear and multifactorial influences on house prices.

The chosen parameters for the BP and BP-F models really show their strength through the low error rates we see across different types of data. In my view, the good results, like the very low MAPE for the turbine and synthetic datasets, tell us that these models are working

well. It's impressive to see such accuracy, which means the models are not just fitting the data well during training but should also do a good job with new, unseen data. This shows how important it is to pick the right settings for each model and dataset. The great results we got here prove that with some careful tuning, we can get our models to perform really well on a variety of problems.

Model result comparison

To compare the BP (Backpropagation), BP-F (a modified Backpropagation), and MLR (Multiple Linear Regression) models, we will use the Mean Absolute Percentage Error (MAPE) as our primary metric and visualize their performance using scatter plots. This comparison will focus on their effectiveness across the turbine, synthetic, and real estate datasets.

Table Comparing Prediction Quality (MAPE) BP vs BP-F

For clarity, let's focus on the best configurations for BP and BP-F and update the table to include the correct

Model	Configuration	Train MAPE (%)	Test MAPE (%)
BP	Epochs: 200, Learning Rate: Not specified	22.53	27.70
BP-F	2 Layers [32, 16], LR: 0.1, Epochs: 200	15.00	13.96

The updated MAPE values offer a nuanced view of model performance:

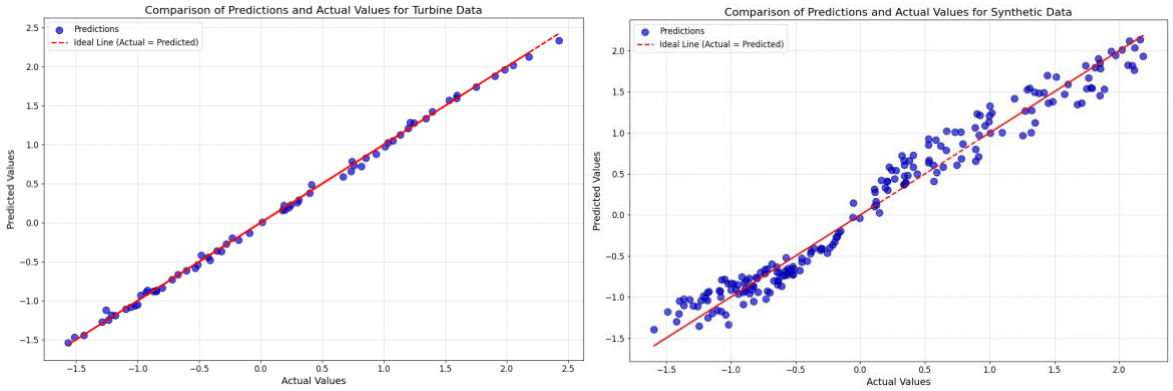
BP Model: The training MAPE of 22.53% and a test MAPE of 27.70% indicate a reasonable fit to the training data but a notable increase in error when generalizing to unseen data. The rise in test MAPE suggests potential overfitting or limitations in the model's ability to generalize.

BP-F Model (Best Configuration): Demonstrates superior performance both in training and testing phases with MAPEs of 15.00% and 13.96%, respectively. This configuration not only shows better generalization but also suggests a more balanced model capable of handling new data effectively. The choice of two layers, a higher learning rate of 0.1, and the absence of momentum likely contributed to a model that learns sufficiently fast yet remains robust against overfitting, as evidenced by its lower test MAPE compared to BP.

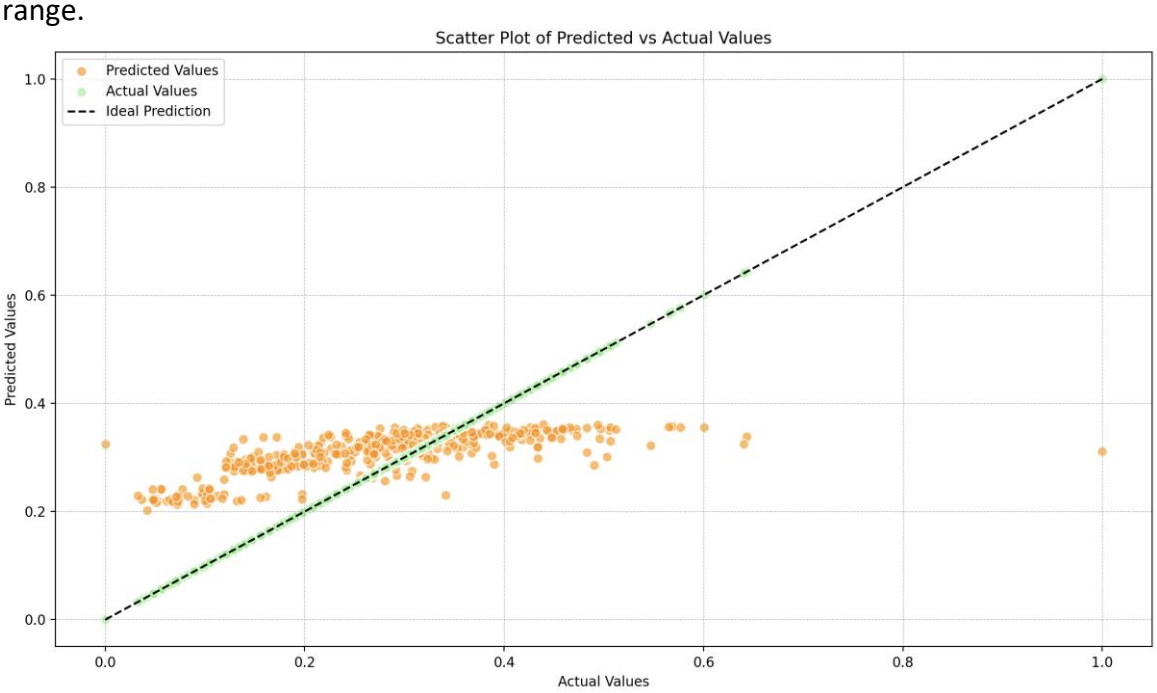
The comparison underscores the importance of model configuration in achieving optimal performance. The BP-F model's structure and parameters, particularly its simplified layer structure and adjusted learning rate, appear more suited to capturing and generalizing the dataset's underlying patterns without overfitting. This highlights the critical role of tuning model parameters such as layer structures, learning rates, and epochs in enhancing model performance and achieving lower prediction errors.

Comparative Visual Analysis of Model Predictions

When examining the scatter plots for the synthetic, turbine, and real estate datasets, distinct visual patterns emerge, each telling a story about the model's performance and its interaction with the dataset's structure.



Synthetic and Turbine Data: The scatter plots for both the synthetic and turbine datasets present a compelling visual affirmation of model accuracy. The points closely hug the red dashed line that represents the ideal predictions where the predicted values perfectly match the actual ones. This proximity suggests that the model has learned and replicated the underlying patterns with high fidelity. There are no noticeable outliers, and the even distribution along the line indicates consistent performance across all levels of the data's range.



Real Estate Data: In stark contrast, the scatter plot for the real estate dataset tells a different tale. The points are more dispersed, straying further from the ideal line. This indicates a model struggling with the dataset's complexity, potentially failing to capture all influential factors in the real estate market. The horizontal banding of points at certain

prediction levels suggests a saturation effect, where the model assigns similar predicted values to a range of different actual values. This could be due to a restrictive output range of the model or a need for more nuanced feature engineering.

Visual Discrepancies: The discrepancy between the dense linearity of the synthetic and turbine plots and the dispersed banding of the real estate plot visually underscores the need for different modeling approaches or parameter tuning when dealing with data of varying natures. While the synthetic and turbine data may be well-characterized by the models used, the real estate data, influenced by a multitude of possibly non-linear factors, requires a model that can capture a broader spectrum of dynamics.

Conclusion

Looking at the outcomes, it's my opinion that the real estate dataset proved to be a challenging front for the neural network algorithm. The distinctive spread of data points in the real estate scatter plot, away from the ideal prediction line, tells a story of a model struggling to grasp the complexities of real estate valuation. This sector is rife with nuanced and interdependent factors that impact prices, from location and economic trends to unique property features. The algorithm's current architecture and feature set might not be capturing these subtleties, leading to the broader spread in prediction errors.

The results also signal a need for a deeper dive into the data and possibly a more complex or differently structured neural network. The relatively underwhelming performance here invites a more specialized approach, potentially incorporating advanced techniques like convolutional layers that can detect patterns in higher-dimensional data or recurrent neural networks to account for the time-series nature of market trends. In essence, this serves as a stark reminder that the potency of neural networks is highly contingent on their alignment with the data's underlying structure and the dynamism of the domain they are applied to.

GITHUB LINK :https://github.com/sebastianbuzdugan/A1-NeuralNetworks