Data-Driven Design Optimization

Jeongwhan Choi

KAIST
Graduate School of Data Science
hwani.choi@kaist.ac.kr

Abstract

This paper presents a study on optimizing data-driven design models using machine learning and deep learning techniques. Initially, i analyze the dataset through descriptive statistics and visualizations to understand the data's characteristics. To enhance model performance and reduce data noise, i focus on sampling subsets of the data with high scores. I employ a range of models from linear regression to complex neural networks to predict and optimize design scores. The study includes models like Random Forest and Support Vector Regression, along with deep learning architectures that use advanced activation functions and regularization methods. The models are evaluated based on their ability to accurately predict design scores, using metrics such as Mean Squared Error (MSE) and R-Square values. my results demonstrate the potential of advanced models to extract meaningful insights from data, particularly when focusing on high-scored segments to improve predictive accuracy and guide future design choices. This approach illustrates the importance of selective data sampling in enhancing the effectiveness of predictive models in data-driven design optimization.

1 Data Description

To understand the characteristics of the data, i first calculated the descriptive statistics and then plotted a scatter plot.

```
Code :
log.describe()
```

Listing 1: Python code to compute and display descriptive statistics of the dataset

	A	**B**	**C**	**D**	**E**	**score**
count	9992.000000	9992.000000	9992.000000	9992.000000	9992.000000	9992.000000
mean	0.498985	0.498149	0.494426	0.495300	0.498725	71.645244
std	0.291844	0.289512	0.289011	0.289082	0.289978	4.333470
min	0.000064	0.000010	0.000397	0.000126	0.000233	62.944380
25%	0.244372	0.247665	0.241093	0.242532	0.245855	68.460217
50%	0.499789	0.498503	0.491682	0.494536	0.497319	70.780107
75%	0.754511	0.749839	0.747202	0.743745	0.750852	74.079871
max	0.999860	0.999969	0.999882	0.999996	0.999987	89.830920

Table 1: Descriptive statistics of the dataset from `log.describe()`

In the scatter plot, i calculated the average values of each column corresponding to the top 0.5% of scores (indicated by the blue dashed line) and represented by the red solid line. This process helps me understand the typical values that each column assumes when the scores are high. These values can be represented in dictionary form as {'A': 0.6707, 'B': 0.5951, 'C': 0.2594, 'D': 0.7433, 'E': 0.3981}. Ordered sequentially, the values are 'D' > 'A' > 'B' > 'E' > 'C'. This order can be used to tentatively predict the importance of each score as i commence the analysis.

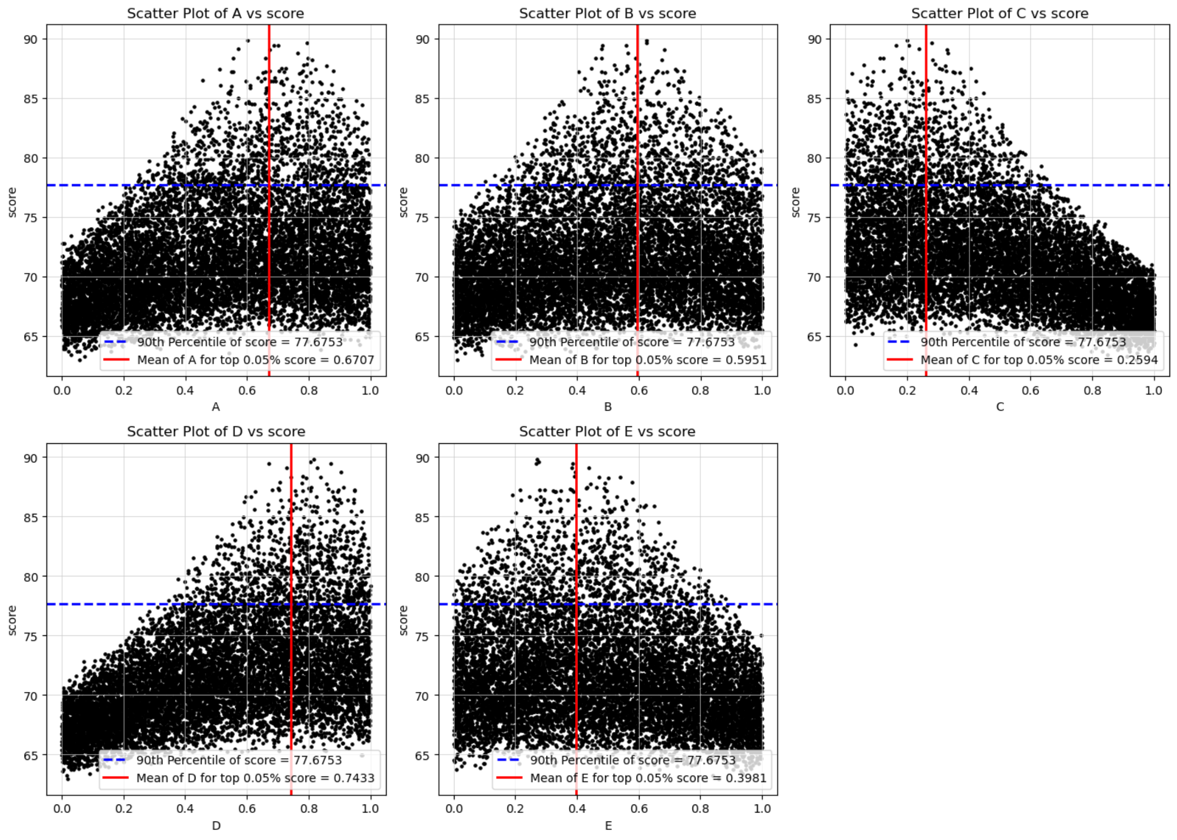


Figure 1: Scatter plot illustrating the distribution of features and scores

Additionally, by plotting the heatmap, i confirmed that the order 'D' > 'A' > 'B' > 'E' > 'C', previously observed in the scatter plot, corresponds to the sequence of columns with the highest correlation with the score.

Given that the goal is to achieve high scores using the combination of 'A', 'B', 'C', 'D', 'E', i aim to train the model using two options. The first option is to use all the raw data available in `log.csv` to train the model. The second option is to train the model using only the data corresponding to the top 10% of scores from `log.csv`. Particularly for the second option, there is an expectation that removing the bottom 90% of data points, which may not show significant relationships and could act as noise, will be advantageous in identifying patterns more effectively.

Ultimately, after training models using both options, the objective is to compare the performance of each model and extract the optimal set of 10 queries.

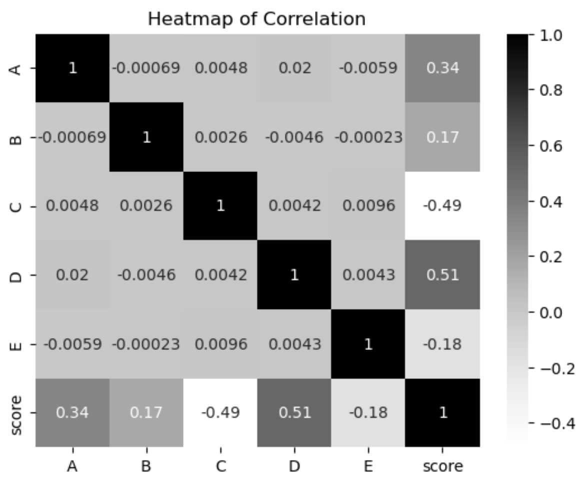


Figure 2: Heatmap indicating the correlation between features and the score

2 Model Selection and Optimization Algorithm

2.1 Method Description

The following ML models were explored to understand their performance in predicting the unknown black-box function:

2.1.1 Machine Learning Models

- **Linear Regression:** A simple model that assumes a linear relationship between the input variables and the target variable. It is used here as a baseline to compare more complex models.
- **Random Forest:** An ensemble learning method that uses multiple decision trees to improve predictive accuracy and control over-fitting. It is known for its high accuracy, robustness, and ease of use.
- **Support Vector Regression (SVR):** An extension of the Support Vector Machine (SVM) that supports linear and non-linear regression. I utilized SVR due to its effectiveness in high-dimensional spaces.

2.1.2 Deep Learning Models

To capture more complex nonlinear relationships in the data, i experimented with the following DL models:

- **Base Neural Network:** This simple neural network model consists of two hidden layers with 64 neurons each, using ReLU activation. The architecture is:

$$[5 \rightarrow 64, \text{ReLU}, 64 \rightarrow 64, \text{ReLU}, 64 \rightarrow 1]$$

- **Improved Neural Network:** To enhance the model's learning capability, i introduced batch normalization and dropout layers to reduce overfitting and improve generalization. The architecture is:

$$[5 \rightarrow 64, \text{ReLU}, \text{BatchNorm}, \text{Dropout}(0.5), 64 \rightarrow 64, \text{ReLU}, \text{BatchNorm}, \text{Dropout}(0.5), 64 \rightarrow 1]$$

- **Complex Neural Network:** Designed to capture deeper patterns in the data, this model increases the network's depth and introduces PReLU activation for non-linearity. The architecture includes:

$$\begin{bmatrix} 5 \to 128, \text{BatchNorm}, \text{PReLU}, \text{Dropout}(0.3), \\ 128 \to 256, \text{BatchNorm}, \text{PReLU}, \text{Dropout}(0.3), \\ 256 \to 64, \text{BatchNorm}, \text{PReLU}, \text{Dropout}(0.3), \\ 64 \to 1 \end{bmatrix}$$

2.2 Optimization Approach

Given the potential complexities of the black-box function, i adopted Bayesian Optimization as my primary strategy for finding optimal input configurations. This approach is particularly suited for situations where the objective function is expensive to evaluate or does not have an analytic form, as is often the case in machine learning and deep learning scenarios.

2.2.1 Bayesian Optimization Overview

Bayesian Optimization is a probabilistic model-based optimization technique used for minimizing or maximizing an unknown objective function. The key steps involved in Bayesian Optimization are:

1. **Define a Prior Over the Function:** The prior belief about the objective function is expressed as a Gaussian Process (GP).
2. **Select a Sampling Criterion:** A function known as the acquisition function is used to sample the next point to evaluate. Common choices include Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB).
3. **Update the Posterior:** Using the samples and the prior, the posterior distribution over the objective function is updated, reflecting new information learned from the evaluations.

2.2.2 Application to Model Selection

For each ML and DL model, excluding Linear Regression due to its simplicity and lack of hyperparameters for meaningful optimization, i applied Bayesian Optimization to identify the set of inputs that maximize the predicted score. This was performed by:

- Training the model on the dataset extracted from `log.csv`.
- Using the trained model within the Bayesian Optimization framework to explore and optimize the input space.
- Extracting and averaging the scores from 10 optimal queries suggested by Bayesian Optimization for each model.

The performance of the models was then compared based on the average score obtained from these optimal points, MSE error, and Adjusted R-Square value. This approach provided a clear metric for model evaluation, offering insights into the underlying black-box function.

Mean Squared Error (MSE) is a common measure of the difference between values predicted by a model and the values actually observed from the environment that is being modeled. It is calculated using the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (1)$$

where y_i is the actual value observed, $\hat{y}_i$ is the predicted value, and n is the total number of observations.

R-Squared ($\mathbf{R^2}$) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. The definition is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2}$$

where y_i is the actual value observed, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the observed data.

Adjusted R-Squared further refines the $\mathrm{R}^{\mathbf{2}}$ by adjusting for the number of explanatory variables in the model and only increases if the new term improves the model more than would be expected by chance. It is calculated as:

$$\text{Adjusted } R^2 = 1 - \left(1 - R^2\right) \frac{n-1}{n-p-1} \tag{3}$$

where n is the number of observations and p is the number of predictors in the model.

Model Score Calculation: The model score was devised using the formula:

$$\text{Model Score} = \text{Adjusted } R^2 \times \left(\frac{2 \times \text{Average Score} \times \left(\frac{1}{\sqrt{MSE}}\right)}{\text{Average Score} + \left(\frac{1}{\sqrt{MSE}}\right)} \right) \tag{4}$$

This formula integrates the Average Score and the inverse square root of MSE, weighted by the $\mathrm{R}^{\mathbf{2}}$ value. The rationale behind this is to balance the predictive accuracy (reflected by the inverse of the MSE's square root) and the model's fit (indicated by Adjusted $\mathrm{R}^{\mathbf{2}}$) while normalizing the contribution of each metric to ensure that neither dominates the score. The harmonic mean approach ensures that both the average score's magnitude and the precision measured by MSE contribute equally to the final score, enhancing the interpretability and fairness of model comparisons.

3 Experiment

Since the code implementing each model is included, details on hyperparameter tuning will be omitted.

3.1 Model Description

3.1.1 Machine Learning

A. Linear Regression In this model, i utilized the data points to train the model without any specific techniques. Linear Regression does not lend itself to extracting optimal points using Bayesian Optimization due to the linear nature of the hyperplane learned by the model. While finding optimal points via Bayesian Optimization was not meaningful, i saved the top 10 queries where the model achieved the highest scores.

```python
print('--------------------------------------------------------------------------------')
print('-------------------------Processing Linear Regression-----------------------')

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_regression, y_regression, test_size=0.2,
    random_state=42)

# Training the linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predicting 'score' for the test set
y_pred_test = linear_model.predict(X_test)

# Predicting 'score' for the entire dataset
predicted_scores = linear_model.predict(X_regression)
X_regression['predicted_score'] = predicted_scores

# Extracting the top 10 rows with the highest predicted 'score'
top_predictions_regression = X_regression.nlargest(10, 'predicted_score')

# Displaying the top 10 predictions
print(top_predictions_regression[['A', 'B', 'C', 'D', 'E', 'predicted_score']])

# Calculating the average of the top 10 predicted 'scores'
average_score = top_predictions_regression['predicted_score'].mean()
Best_score[f'Linear Regression for {label}'] = average_score
Best_query[f'Linear Regression for {label}'] = top_predictions_regression[['A', 'B', 'C', 'D', 'E',
     'predicted_score']]

print(f"\nAverage score of top 10 predictions using Linear Regression for {label} :", average_score
    )

# Calculating MSE and Adjusted R^2 for the test set predictions
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
r2 = 1 - (1 - r2) * (n_1 - 1) / (n_1 - p - 1) # Calculate Adjusted R^2

print("Mean Squared Error (MSE) on Test Set :", mse)
print("Adujsted R^2 Score on Test Set :", r2)
model_score = r2 * (2 * average_score * (1 / np.sqrt(mse))) / (average_score + (1 / np.sqrt(mse)))
print(f'Model Score : {model_score}')
Model_score[f'Linear Regression for {label}'] = model_score

print('-----------------------Linear Regression Process Done-----------------------')
print('----------------------------------------------------------------------------\n\n')
```

Listing 2: Linear Regression Training Code

Raw data results:

```
----------------------------------------------------------------------------
-------------------------Processing Linear Regression-----------------------
             A         B         C         D         E  predicted_score
1180  0.956516  0.960953  0.024313  0.948932  0.198394        82.850585
2299  0.970314  0.967356  0.020965  0.903119  0.184134        82.648720
7752  0.936599  0.914324  0.105209  0.972479  0.180965        82.256941
2111  0.999785  0.754854  0.061814  0.822084  0.038836        81.699589
3198  0.869196  0.948915  0.148865  0.967087  0.167834        81.691115
8084  0.929628  0.838104  0.110895  0.849916  0.017940        81.483623
6470  0.793848  0.994590  0.137328  0.940213  0.122925        81.446826
9373  0.925830  0.793319  0.122436  0.899091  0.110947        81.384842
5106  0.925213  0.446800  0.043257  0.913535  0.032051        81.371713
603   0.592025  0.917460  0.002837  0.983172  0.245124        81.252973

Average score of top 10 predictions using Linear Regression for Raw data : 81.8087
Mean Squared Error (MSE) on Test Set : 6.2175
Adujsted R^2 Score on Test Set : 0.6710
Model Score : 0.5355
-----------------------Linear Regression Process Done-----------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

```
--------------------------------------------------------------------------
-------------------------Processing Linear Regression-----------------------
             A          B          C          D          E  predicted_score
7867  0.990510  0.741040  0.125927  0.958982  0.509854        82.568007
5723  0.894452  0.878624  0.077888  0.964411  0.623175        82.496247
187   0.802315  0.828741  0.008191  0.972973  0.561132        82.493080
5417  0.812533  0.742994  0.023858  0.979701  0.412444        82.476315
7752  0.936599  0.914324  0.105209  0.972479  0.180965        82.452985
5106  0.925213  0.446800  0.043257  0.913535  0.032051        82.447241
6426  0.970249  0.421977  0.027746  0.820723  0.637021        82.432331
3584  0.971169  0.542276  0.150233  0.927966  0.718677        82.426432
9231  0.920868  0.281228  0.032551  0.865099  0.466281        82.424248
8120  0.926459  0.623024  0.135016  0.976300  0.272531        82.414106

Average score of top 10 predictions using Linear Regression for Top 10% data : 82.4631
Mean Squared Error (MSE) on Test Set : 5.2373
Adjusted R^2 Score on Test Set : 0.0888
Model Score : 0.0772
-----------------------Linear Regression Process Done-----------------------
--------------------------------------------------------------------------
```

B. Random Forest Random Forest is an ensemble model that uses multiple decision trees to improve the prediction performance and reduce overfitting. This model is better suited for use with Bayesian Optimization to find optimal input configurations.

```python
print('--------------------------------------------------------------------------')
print('---------------------------Processing Random Forest-------------------------')

# Assume the training data (X_train, y_train) is already defined from previous context
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predicting 'score' for the test set using the trained Random Forest model
y_pred_test_rf = rf_model.predict(X_test)

# Calculating MSE and Adjusted R^2 for the test set predictions
mse_rf = mean_squared_error(y_test, y_pred_test_rf)
r2_rf = r2_score(y_test, y_pred_test_rf)
r2_rf = 1 - (1 - r2_rf) * (n_1 - 1) / (n_1 - p - 1)

# Objective function to maximize using Bayesian Optimization
def rf_score(A, B, C, D, E):
    X_new = [[A, B, C, D, E]]
    return rf_model.predict(X_new)[0]

# Parameter bounds for Bayesian Optimization
pbounds = {'A': (0, 1), 'B': (0, 1), 'C': (0, 1), 'D': (0, 1), 'E': (0, 1)}

print('------------Finding Optimal Points by using Bayesian Optimization------------')

# Bayesian Optimization object creation
optimizer = BayesianOptimization(
    f=rf_score,
    pbounds=pbounds,
    verbose=1,
    random_state=42
)

# Optimization execution
optimizer.maximize(
    init_points=5,
    n_iter=20
)

# Extracting the top 10 results based on the target (predicted score)
top_results = sorted(optimizer.res, key=lambda x: x['target'], reverse=True)[:10]

# Preparing the DataFrame to display the top results
df = {"A": [], "B": [], "C": [], "D": [], "E": [], "score": []}

for result in top_results:
    df['A'].append(result['params']['A'])
    df['B'].append(result['params']['B'])
    df['C'].append(result['params']['C'])
    df['D'].append(result['params']['D'])
    df['E'].append(result['params']['E'])
    df['score'].append(result['target'])

top_predictions_rf = pd.DataFrame(df)
print(top_predictions_rf)

# Calculating the average score of the top 10 predictions
average_score = np.mean(top_predictions_rf['score'])
Best_score['Random Forest'] = average_score
Best_query['Random Forest'] = top_predictions_rf

print("\nAverage score of top 10 predictions using Random Forest: ", average_score)
print(f"Mean Squared Error (MSE) on Test Set: {mse_rf}")
print(f"R^2 Score on Test Set: {r2_rf}")
model_score = r2_rf * (2 * average_score * (1 / np.sqrt(mse_rf))) / (average_score + (1 / np.sqrt(
    mse_rf)))
print(f'Model Score : {model_score}')

print('-------------------------Random Forest Process Done-------------------------')
print('----------------------------------------------------------------------------')
```

Listing 3: Random Forest Training Code

Raw data results:

```
----------------------------------------------------------------------------
---------------------------Processing Random Forest-------------------------
          A         B         C         D         E     score
0  0.803046  0.692764  0.346689  0.639745  0.535995  88.31569
1  0.779011  0.735261  0.217099  0.697288  0.565222  87.84631
2  0.580902  0.690155  0.029076  0.866828  0.426997  87.39833
3  0.738339  0.503028  0.000000  0.858398  0.448812  87.39833
4  0.708950  0.627586  0.129028  0.805071  0.542781  87.39833
5  0.643046  0.592230  0.000000  0.814743  0.515416  87.39833
6  0.675312  0.627858  0.074812  0.887660  0.459394  87.39833
7  0.794941  0.722447  0.156531  0.872999  0.514544  87.39833
8  0.774567  0.646938  0.000000  0.833239  0.580494  87.39833
9  0.680550  0.715370  0.183278  0.895488  0.544390  87.39833

Average score of top 10 predictions using Random Forest for Raw data:  87.5349
Mean Squared Error (MSE) on Test Set: 1.6323
Adjusted R^2 Score on Test Set: 0.9136
Model Score : 1.4175
-------------------------Random Forest Process Done-------------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

```
----------------------------------------------------------------------------
---------------------------Processing Random Forest-------------------------
          A         B         C         D         E     score
0  0.620983  0.589436  0.109792  0.793565  0.433967  89.83092
1  0.611822  0.549189  0.177204  0.731520  0.432284  89.83092
2  0.614612  0.504475  0.162490  0.749267  0.297567  89.83092
3  0.623226  0.525527  0.183841  0.755162  0.292917  89.83092
4  0.677147  0.586609  0.214032  1.000000  0.364515  89.41499
5  0.713916  0.594703  0.184491  0.828604  0.374964  89.41499
6  0.684432  0.508711  0.242458  0.896791  0.478080  89.41499
7  0.698600  0.538272  0.148396  0.859429  0.465164  89.41499
8  0.677909  0.567423  0.092613  0.723121  0.413857  89.41499
9  0.696296  0.511413  0.124496  0.801471  0.252995  89.41499

Average score of top 10 predictions using Random Forest for Top 10% data:  89.5814
Mean Squared Error (MSE) on Test Set: 3.7863
Adjusted R^2 Score on Test Set: 0.3413
Model Score : 0.3488
-------------------------Random Forest Process Done-------------------------
----------------------------------------------------------------------------
```

C. Support Vector Regression (SVR) Support Vector Regression extends SVM to support regression, providing effective high-dimensional pattern recognition.

```
print('----------------------------------------------------------------------------')
print('---------------------Processing Support Vector Regression-------------------')

```

```
# Training the Support Vector Regression (SVR) model with chosen parameters
model = SVR(kernel='rbf', C=195, epsilon=0.1)
model.fit(X_train, y_train)

# Predicting 'score' for the test set using the trained SVR model
y_pred_test_svr = model.predict(X_test)

# Objective function to maximize using Bayesian Optimization
def svr_score(A, B, C, D, E):
    X_new = [[A, B, C, D, E]]
    return model.predict(X_new)[0]

# Parameter bounds for Bayesian Optimization
pbounds = {'A': (0, 1), 'B': (0, 1), 'C': (0, 1), 'D': (0, 1), 'E': (0, 1)}

print('------------Finding Optimal Points by using Bayesian Optimization------------')

# Bayesian Optimization object creation
optimizer = BayesianOptimization(
    f=svr_score,
    pbounds=pbounds,
    verbose=1,
    random_state=42
)

# Optimization execution
optimizer.maximize(
    init_points=5,
    n_iter=20
)

# Extracting the top 10 results based on the target (predicted score)
top_results = sorted(optimizer.res, key=lambda x: x['target'], reverse=True)[:10]

# Preparing the DataFrame to display the top results
df = {"A": [], "B": [], "C": [], "D": [], "E": [], "score": []}

for result in top_results:
    df['A'].append(result['params']['A'])
    df['B'].append(result['params']['B'])
    df['C'].append(result['params']['C'])
    df['D'].append(result['params']['D'])
    df['E'].append(result['params']['E'])
    df['score'].append(result['target'])

top_predictions_bayes = pd.DataFrame(df)
print(top_predictions_bayes)

# Calculating the average score of the top 10 predictions
average_score = np.mean(top_predictions_bayes['score'])
Best_score['Bayesian Optimization'] = average_score
Best_query['Bayesian Optimization'] = top_predictions_bayes

print(f"Average score of top 10 predictions using Bayesian Optimization: {average_score}")

# Calculating MSE and Adjusted R^2 for the test set predictions
mse_svr = mean_squared_error(y_test, y_pred_test_svr)
r2_svr = r2_score(y_test, y_pred_test_svr)
r2_svr = 1 - (1 - r2_svr) * (n_1 - 1) / (n_1 - p - 1)

print(f"Mean Squared Error (MSE) on Test Set: {mse_svr}")
print(f"R^2 Score on Test Set: {r2_svr}")
model_score = r2_svr * (2 * average_score * (1 / np.sqrt(mse_svr))) / (average_score + (1 / np.sqrt
    (mse_svr)))
print(f'Model Score : {model_score}')

print('-------------------Support Vector Regression Process Done-------------------')
print('----------------------------------------------------------------------------')
```

Listing 4: Support Vector Regression Training Code

Raw data results:

```
----------------------------------------------------------------------------
---------------------Processing Support Vector Regression-------------------
          A         B         C         D         E      score
0  0.706671  0.601959  0.189755  0.808141  0.395875  90.864671
1  0.705952  0.603603  0.190062  0.807068  0.395576  90.864637
2  0.707849  0.602194  0.191299  0.807275  0.393856  90.864590
3  0.707599  0.602314  0.190871  0.807080  0.393432  90.864587
4  0.707523  0.602329  0.190935  0.807155  0.393394  90.864585
5  0.708501  0.603299  0.191121  0.807637  0.394482  90.864577
```

```
6  0.707640  0.602142  0.191116  0.807342  0.393503  90.864572
7  0.707043  0.602087  0.189842  0.805984  0.394651  90.864568
8  0.707717  0.602393  0.191128  0.807191  0.393476  90.864566
9  0.707061  0.601507  0.189951  0.806340  0.394367  90.864556
Average score of top 10 predictions using SVR for Raw data: 90.8646
Mean Squared Error (MSE) on Test Set: 0.1300
Adjusted R^2 Score on Test Set: 0.9931
Model Score : 5.3457
-------------------Support Vector Regression Process Done-------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

```

----------------------------------------------------------------------------
---------------------Processing Support Vector Regression-------------------
          A         B         C         D         E      score
0  0.696940  0.598478  0.207432  0.800064  0.396308  91.982319
1  0.697148  0.598605  0.207406  0.800035  0.396359  91.982301
2  0.697284  0.597336  0.206357  0.799729  0.395881  91.982273
3  0.697419  0.598703  0.207437  0.799969  0.396389  91.982271
4  0.697402  0.597213  0.206410  0.799651  0.395967  91.982269
5  0.697361  0.597263  0.206391  0.799636  0.395935  91.982266
6  0.697360  0.597246  0.206299  0.799690  0.396053  91.982262
7  0.697334  0.597224  0.206297  0.799621  0.395893  91.982239
8  0.697387  0.597341  0.206317  0.799460  0.395907  91.982228
9  0.697630  0.598846  0.207443  0.799884  0.396443  91.982215
Average score of top 10 predictions using SVR for Top 10% data: 91.9823
Mean Squared Error (MSE) on Test Set: 0.1732
Adjusted R^2 Score on Test Set: 0.9699
Model Score : 4.5418
-------------------Support Vector Regression Process Done-------------------
----------------------------------------------------------------------------
```

3.1.2 Deep Learning

A. Base Neural Network A simple neural network model with two hidden layers, each consisting of 64 neurons and ReLU activation. This is a foundational model to establish a baseline for more complex architectures.

```python
# Define a Simple Neural Network
class BaseMLP(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim):
        super(BaseMLP, self).__init__()

        self.model = nn.Sequential(nn.Linear(input_dim, hidden_dim),
                                   nn.ReLU(),
                                   nn.Linear(hidden_dim, hidden_dim),
                                   nn.ReLU(),
                                   nn.Linear(hidden_dim, output_dim))

    def forward(x):
        return self.model(x)
```

Listing 5: Base Neural Network Training Code

Raw data results:

```


-----------------------------Experiment setting------------------------------

Batch size : 256
Training Step : 50000
Learning rate : 0.0001

          A         B    C    D         E      score
0  0.845392  0.628553  0.0  1.0  0.427318  89.470619
1  0.845089  0.628645  0.0  1.0  0.426754  89.467964
2  0.844859  0.628704  0.0  1.0  0.426269  89.465523
3  0.845927  0.628919  0.0  1.0  0.425877  89.464149
4  0.844630  0.628770  0.0  1.0  0.425868  89.463371
5  0.844423  0.628806  0.0  1.0  0.425491  89.461494
```

```
6  0.844196  0.628835  0.0  1.0  0.425141  89.459717
7  0.845826  0.628470  0.0  1.0  0.428049  89.459435
8  0.843996  0.628865  0.0  1.0  0.424825  89.458115
9  0.845907  0.628434  0.0  1.0  0.428195  89.456978

Average score of top 10 predictions using Base Neural Network for Raw data :  86.4871
Mean Squared Error (MSE) on Test Set : 0.9085
Adjusted R^2 Score on Test Set : 0.9519
Model Score : 1.9734
---------------------Base Deep learning Process Done---------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

```

-----------------------------Experiment setting------------------------------
Batch size : 256
Training Step : 50000
Learning rate : 0.0001

          A         B         C         D         E      score
0  0.722444  0.567663  0.000000  1.000000  0.382730  88.251740
1  0.714152  0.562271  0.000000  1.000000  0.364230  88.091324
2  0.723264  0.593006  0.000000  1.000000  0.387052  88.069054
3  0.713897  0.589226  0.000000  0.964109  0.385616  87.987434
4  0.748668  0.568674  0.000000  1.000000  0.364642  87.918015
5  0.727816  0.549274  0.000000  1.000000  0.398851  87.887199
6  0.716534  0.593926  0.025074  1.000000  0.386334  87.886177
7  0.759538  0.567589  0.042632  1.000000  0.364771  87.756889
8  0.737440  0.538021  0.074656  1.000000  0.385252  87.706360
9  0.684546  0.556242  0.000000  1.000000  0.385631  87.692772

Average score of top 10 predictions using Base Neural Network for Top 10% data:  87.9247
Mean Squared Error (MSE) on Test Set : 2.7297
Adjusted R^2 Score on Test Set : 0.5251
Model Score : 0.6313
```

B. Improved Neural Network This enhanced version of the base model includes batch normalization and dropout layers to improve generalization and prevent overfitting.

```python
class ImprovedMLP(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dim):
        super(ImprovedMLP, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim),
            nn.Dropout(0.5),

            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.BatchNorm1d(hidden_dim),
            nn.Dropout(0.5),

            nn.Linear(hidden_dim, output_dim)
        )

    def forward(x):
        return self.model(x)
```
Listing 6: Improved Neural Network Training Code

Raw data results:

```

-----------------------------Experiment setting------------------------------
Batch size : 256
Training Step : 500000
Learning rate : 0.0001

          A         B         C    D         E      score
0  0.862167  0.801411  0.058822  1.0  0.354710  83.681747
1  0.870010  0.799465  0.034410  1.0  0.389043  83.679024
```

```
2  0.891658  0.805696  0.000000  1.0  0.355497  83.678757
3  0.871941  0.777389  0.049860  1.0  0.358451  83.676888
4  0.856632  0.806154  0.057933  1.0  0.457362  83.676453
5  0.885766  0.786405  0.035910  1.0  0.324713  83.676346
6  0.872403  0.789659  0.061746  1.0  0.491845  83.673309
7  0.880800  0.816326  0.000000  1.0  0.402335  83.672394
8  0.856978  0.788900  0.055303  1.0  0.321386  83.669456
9  0.878668  0.811893  0.018604  1.0  0.468327  83.669067

Average score of top 10 predictions using Improved Neural Network for Raw data :  83.6754
Mean Squared Error (MSE) on Test Set : 1.2905
Adjusted R^2 Score on Test Set : 0.9317
Model Score : 1.6233
---------------------Improved Deep learning Process Done---------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

```

-----------------------------Experiment setting------------------------------
Batch size : 256
Training Step : 500000
Learning rate : 0.0001

          A         B         C         D         E      score
0  0.696106  0.502166  0.273734  1.000000  0.438207  85.505714
1  0.664210  0.530377  0.361546  0.920443  0.447591  85.438110
2  0.591037  0.510064  0.372535  0.847966  0.388775  85.423965
3  0.731008  0.603139  0.212863  1.000000  0.346167  85.412590
4  0.708513  0.603272  0.129381  1.000000  0.275756  85.407242
5  0.584943  0.521285  0.296269  0.905787  0.422866  85.374855
6  0.756916  0.595668  0.287757  1.000000  0.439914  85.346336
7  0.580958  0.528887  0.368327  0.891948  0.471256  85.345444
8  0.670541  0.511938  0.231762  0.946889  0.457988  85.318130
9  0.634813  0.581336  0.141997  1.000000  0.334753  85.318047

Average score of top 10 predictions using Improved Neural Network for Top 10% data :  85.3890
Mean Squared Error (MSE) on Test Set : 2.0014
Adjusted R^2 Score on Test Set : 0.6518
Model Score : 0.9139
---------------------Improved Deep learning Process Done---------------------
----------------------------------------------------------------------------
```

C. Complex Neural Network A more complex neural network that incorporates deeper layers and PReLU activation to capture more intricate patterns in the data.

```python
class ComplexMLP(nn.Module):
    def __init__(self, input_dim, output_dim, hidden_dims):
        super(ComplexMLP, self).__init__()

        layers = []
        prev_dim = input_dim

        for hidden_dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, hidden_dim))
            layers.append(nn.BatchNorm1d(hidden_dim))
            layers.append(nn.PReLU())
            layers.append(nn.Dropout(0.3))
            prev_dim = hidden_dim

        layers.append(nn.Linear(hidden_dims[-1], output_dim))

        self.model = nn.Sequential(*layers)

    def forward(x):
        return self.model(x)
```

Listing 7: Complex Neural Network Training Code

Raw data results:

```

-----------------------------Experiment setting------------------------------
Batch size : 256
```

```
Training Step : 500000
Learning rate : 0.0001

          A         B         C         D         E      score
0  0.606358  0.553522  0.097597  0.870574  0.295415  86.733086
1  0.601654  0.686506  0.150346  0.890788  0.393796  86.660767
2  0.601656  0.554600  0.114553  0.877169  0.365497  86.652603
3  0.619993  0.688901  0.131671  0.844494  0.328561  86.644997
4  0.744635  0.706688  0.234646  0.860644  0.255769  86.610756
5  0.665100  0.685550  0.131243  0.802577  0.286205  86.603577
6  0.608646  0.671974  0.213327  0.942166  0.402556  86.600433
7  0.648610  0.562557  0.123223  0.869861  0.502642  86.598946
8  0.637683  0.621511  0.163660  0.905064  0.527574  86.577026
9  0.666475  0.682581  0.167111  0.881285  0.504764  86.573143

Average score of top 10 predictions using Complex Neural Network for Raw data :  86.6256
Mean Squared Error (MSE) on Test Set : 0.8371
Adjusted R^2 Score on Test Set : 0.9557
Model Score : 2.0631
---------------------Complex Deep learning Process Done---------------------
----------------------------------------------------------------------------
```

Top 10% Score data results:

No Result(Insuffient data points to train the Complex model)

3.2 Experiment Analysis

3.2.1 The Best Model: Support Vector Regression(SVR)

As can be seen from the results, the Support Vector Regression (SVR) model yields the best performance when estimating the black-box function. This superior performance can be attributed to several factors:

- **Robustness to Outliers:** SVR is known for its robustness to outliers, which allows it to effectively ignore or diminish the noise in the data, focusing more on the main patterns that relate to higher scores.
- **Effective in High Dimensionality:** Despite the data initially being in a five-dimensional space ('A', 'B', 'C', 'D', 'E'), SVR handles high-dimensional spaces well, especially using kernel tricks to map inputs into higher-dimensional spaces where linear relationships can be found.
- **Flexibility:** The SVR model can model non-linear relationships thanks to the kernel trick, and this flexibility allows it to capture complex patterns that might be missed by simpler linear models.
- **Generalization:** SVR tends to have good generalization capabilities, which prevents overfitting by ensuring that the model does not become too complex. This is controlled through the regularization parameter, which has likely been tuned well in this scenario.

By focusing on these significant patterns and effectively managing data complexity and noise, SVR outperforms other models in predicting the score associated with the black-box function.

3.2.2 Top 10% Score Dataset

Training the model using only the top 10% of the data, based on the score, has led to improved performance. This improvement can be attributed to the lack of a distinct pattern in the lower score distribution, where the data points are widely scattered. In other words, the bottom 90% of data points, based on the score, can act as a form of noise when training the model. Hence, the performance is likely better when the model is trained after removing these bottom 90% of data points.

Below is a table comparing the performance of each model using the raw data and the Top 10% data:

Model	Perf. on Raw Data	Perf. on Top 10% Data	Improvement
Linear Regression	81.8087	82.4631	0.80%
Random Forest	87.5349	89.5814	2.34%
Support Vector Regression (SVR)	90.8646	91.9823	1.23%
Base Neural Network	86.4627	87.9247	1.66%
Improved Neural Network	83.6754	85.3890	2.05%
Complex Neural Network	86.6256	—	—

Table 2: Performance comparison of models on raw and top 10% score data with improvements

As demonstrated in Table 2, training models using only the top 10% of the data significantly contributed to increasing the maximum score. There was an improvement in all models, with the Random Forest model showing a notable performance enhancement of approximately 2.34%. Among the models analyzed, the SVR exhibited the highest performance, especially the SVR trained on the top 10% of scores, which achieved an impressive score of 91.9823.

In the case of the Complex Neural Network model, training exclusively on the top 10% of the data led to insufficient data points, which prevented the model from learning effectively. Consequently, during the process of finding the optimal query using Bayesian Optimization, duplicate values within the bounds were encountered, rendering the performance unverifiable.

Model	Score on Raw Data
Linear Regression	0.5355
Random Forest	1.4175
Support Vector Regression (SVR)	5.3457
Base Neural Network	1.3557
Improved Neural Network	1.6233
Complex Neural Network	2.0631

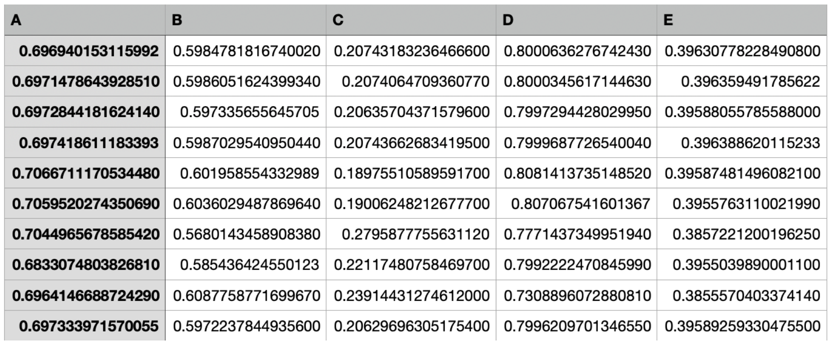
Table 3: Model scores for raw data

Based on the proprietary Model score introduced earlier, the performance of each model trained on raw data was compared. According to the Model score, the SVR demonstrated the highest performance with a score of 5.3457. The Complex Neural Network followed with a score of 2.0631, showing commendable performance after the SVR but falling short due to its inadequate training on the top 10% of the data. Additionally, there is a significant disparity between the top-performing model and the second-best. This gap allows us to conclude that the SVR is overwhelmingly superior compared to the other models.

3.2.3 Selecting Queries

Based on the analyses, 10 queries were selected as follows:

The reasons for selecting the queries are as follows. First, by examining the model scores, i can see that the SVR model performed the best. Accordingly, 2 queries were selected from the best-performing model(trained on Raw data), SVR. Additionally, when the SVR model used the top 10% data instead of the raw data, there was an improvement in the maximum score. Based on this observation, another



A	B	C	D	E
0.696940153115992	0.5984781816740020	0.20743183236466600	0.8000636276742430	0.39630778228490800
0.6971478643928510	0.5986051624399340	0.2074064709360770	0.8000345617144630	0.396359491785622
0.6972844181624140	0.597335655645705	0.20635704371579600	0.7997294428029950	0.39588055785588000
0.697418611183393	0.5987029540950440	0.20743662683419500	0.7999687726540040	0.396388620115233
0.7066711170534480	0.601958554332989	0.18975510589591700	0.8081413735148520	0.39587481496082100
0.7059520274350690	0.6036029487869640	0.19006248212677700	0.807067541601367	0.3955763110021990
0.7044965678585420	0.5680143458908380	0.2795877755631120	0.7771437349951940	0.3857221200196250
0.6833074803826810	0.585436424550123	0.22117480758469700	0.7992222470845990	0.3955039890001100
0.6964146688724290	0.6087758771699670	0.23914431274612000	0.7308896072880810	0.3855570403374140
0.697333971570055	0.5972237844935600	0.20629696305175400	0.7996209701346550	0.39589259330475500

Figure 3: Fianl Query

4 queries were extracted from the SVR trained on the top 10% data. Therefore, a total of 6 queries were extracted from the SVR model.

For similar reasons, 2 query was extracted from the Random Forest model using raw data and 2 query was extracted from the Random Forest trained on the top 10% data.

Lastly, from the Data Description Section, it is observed that the statistically computed values are quite similar to those derived from the models. Additionally, from the heatmap results, it is clear that the importance of the features follows the order $'D' > 'A' > 'B' > 'E' > 'C'$. This ordering is well-reflected in the scatter plot.

4 Conclusion

This study demonstrated the effectiveness of various machine learning and deep learning models in optimizing and predicting design scores from complex datasets. My initial analysis, involving descriptive statistics and data visualizations, revealed critical insights into the feature distributions and relationships within the data.

Among the models evaluated, Support Vector Regression (SVR) stood out due to its robustness in handling high-dimensional spaces and modeling non-linear relationships. This highlights SVR's capability to manage complex, real-world datasets where traditional linear models might underperform. Bayesian Optimization was crucial in identifying optimal input configurations for my models. This

method enhanced my ability to explore the parameter space efficiently, leading to the discovery of optimal points that maximize the predictive performance of my models. My experiments with various

neural network architectures showed that deep learning could capture intricate patterns in the data that simpler models might miss. The Improved Neural Network model, with batch normalization and dropout layers, was particularly effective in reducing overfitting and improving the generalizability of my predictions. Key takeaways from this research include:

- The effectiveness of SVR in complex predictive tasks.
- The utility of Bayesian Optimization in identifying optimal model inputs.
- The potential of deep learning models to outperform simpler baseline models in complex scenarios.

Future work could explore more diverse datasets, apply these models to other domains such as healthcare, finance, and urban planning, and experiment with more complex neural network architectures to further validate and extend my findings. In conclusion, this work provides a robust framework for using advanced machine learning techniques to enhance data-driven design and decision-making processes.

References

[1] Cortes, C., & Vapnik, V. (1995). *Support-Vector Networks.* Machine Learning, 20(3), 273–297. https://doi.org/10.1007/BF00994018

[2] Breiman, L. (2001). *Random Forests.* Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[3] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning.* Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

[4] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016). *Taking the Human Out of the Loop: A Review of Bayesian Optimization.* Proceedings of the IEEE, 104(1), 148–175. https://doi.org/10.1109/JPROC.2015.2494218

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press. http://www.deeplearningbook.org

[6] Freedman, D. A. (2009). *Statistical Models: Theory and Practice.* Cambridge University Press.

[7] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.

[8] Chollet, F., & others. (2015). *Keras.* https://keras.io

[9] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* Advances in Neural Information Processing Systems 32. https://papers.neurips.cc/paper/2019/hash/9015d85cc55a92e034e76f7a5880aa9f-Abstract.html