POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Analysis Project

Author(s): **Lorenzo Ferretti**

Date: 12/09/2024

Academic Year: 2023-2024

Contents

1 | Introduction

Electricity price forecasting is critically important for energy producers, distributors, and consumers, as accurate predictions can enhance bidding strategies, optimize contract negotiations, and improve risk management. However, electricity prices are influenced by a diverse array of factors, including weather patterns, fuel costs, regulatory changes, and market behaviors, making forecasting a highly challenging problem in the energy sector.

The paper under analysis addresses these challenges by proposing a machine learning technique designed to improve forecasting accuracy. The authors tested various genetic programming (GP) techniques and compared their performance with other standard solutions.

To rigorously analyze the paper, I will address the discussion as follows:

- **Energy Prediction Problem**: I will explain the problem of energy price forecasting, highlighting why the approach I used to address it reflects the methodology employed by the authors of the paper.
- **Theoretical Foundations**: I will explain and analyze the specific characteristics of the proposed techniques in detail, referencing related works from the same author for further context.
- **Data**: I will explain the approach used to create an experiment under conditions that aim to solve the same problem, analyzing which data was used and why it represents a valid choice to reproduce and test these algorithms.
- **Experiments**: I will describe how I implemented and tested different algorithms for the task, along with the methodology applied.
- **Results**: I will present the results obtained from both the genetic programming algorithms and more standard techniques, offering an evaluation of their performance.
- **Conclusion**: A final reflection on the work carried out and its significance.

2 | Energy Price Prediction

The energy prediction problem, particularly related to electricity prices, is critical due to the complexity and dynamic nature of electricity markets. Accurate price forecasting is essential for market participants—generators, distributors, and consumers—who depend on precise predictions to make well-informed decisions. The deregulation of electricity markets has intensified competition and uncertainty, making accurate forecasts crucial for optimizing bidding strategies, contract negotiations, and revenue management. Inaccurate forecasts can lead to significant financial losses due to under- or overestimation of prices.

Electricity markets are influenced by a variety of interconnected factors, such as fuel prices, weather conditions, and CO2 emissions. In the paper, the approach used focuses on feature-based learning without performing time series analysis. This is feasible because energy prices are heavily influenced by several other variables. To maintain consistency with the original study, I too predicted energy prices using only features and not time series data. Additionally, the paper focuses on predicting energy prices 24 hours ahead, and I have followed the same approach to keep the problem as closely aligned with the original as possible.

Although I utilized observations at 6 AM, 12 PM, and 6 PM each day, I still predicted the energy prices for the following 24 hours using only the data from the previous day. For instance, I predicted the price at 12 PM on 23/10/2015, knowing only the conditions from 22/10/2015. Of course, better predictive results could have been achieved by incorporating more past data, but I chose not to make this adjustment in order to remain as close as possible to the conditions of the original paper.

3 | Theory Foundation

3.0.1. GP

Genetic Programming (GP) is a method for evolving computer programs to solve problems. It involves creating a population of candidate solutions, which are represented as tree structures, and evolving them over multiple generations to optimize their performance.

Key Terminology

- **Chromosome**: In Genetic Programming, a chromosome represents a potential solution to a problem. It is typically structured as a tree where nodes can be functions (such as arithmetic operators) or terminals (such as constants or variables). The tree structure allows for the representation of complex expressions or algorithms.
- **Genes**: The individual elements of a chromosome tree. Genes include functions and terminals. Functions are internal nodes in the tree that perform operations (e.g., addition, multiplication), while terminals are leaf nodes representing variables or constants.
- **Population**: A set of chromosomes. The population is a collection of different potential solutions, each represented as a tree. Over successive generations, the population evolves as trees are modified through genetic operations to improve their performance on the given task.
- **Fitness Function**: A function that assesses how well a chromosome performs the desired task. It evaluates the quality of the solution represented by the tree structure. The fitness function typically measures how closely the output of the tree matches the desired outcome or how well it solves the problem.
- **Selection**: The process of choosing chromosomes from the current population to create the next generation. Chromosomes are selected based on their fitness scores, with higher fitness generally increasing the likelihood of selection.
- **Crossover (Recombination)**: An operation where two parent chromosomes (trees)

are combined to produce offspring. This involves exchanging subtrees between the parents. For example, a subtree from one parent might be swapped with a subtree from the other parent, creating new tree structures that inherit parts of both parents.

- **Mutation**: An operation that randomly alters parts of a chromosome (tree) to introduce variability. This could involve changing a function node to a different function or replacing a subtree with a new randomly generated subtree. Mutation helps to maintain genetic diversity within the population and allows exploration of new areas in the solution space.
- **Offspring**: New chromosomes (trees) created from the crossover and mutation operations. These offspring are evaluated and added to the population for the next generation. They inherit characteristics from their parent chromosomes but may also have new features introduced through mutation.
- **Generation**: A complete cycle of the evolutionary process, starting from the initial population of chromosomes to the creation of the next generation. Each generation involves selection, crossover, mutation, and replacement, ultimately leading to improved solutions over successive iterations.

The algorithm

- **Generate Initial Population**: Randomly generate an initial set of chromosomes represented as trees. These trees are constructed from a set of functions and terminals, forming diverse potential solutions.
- **Calculate Fitness**: Evaluate each chromosome (tree) using a fitness function. This function assesses how well the tree solves the problem or performs the task.
- **Select Parents**: Choose trees from the current population based on their fitness. Higher fitness increases the likelihood of a tree being selected as a parent for the next generation.
- **Perform Crossover**: Exchange subtrees between pairs of parent trees to create new offspring. This operation combines features from two parent trees, potentially leading to improved solutions.
- **Perform Mutation**: Introduce random changes to a tree. This can involve altering nodes or replacing subtrees, adding variability to the population and exploring new solution spaces.
- **Form New Population**: Create a new population of trees by replacing old trees

with new offspring. Replacement strategies can vary, such as replacing the entire population or mixing new offspring with some of the old trees.

- **Check Termination Condition**: Determine when to stop the evolutionary process. Termination conditions might include reaching a fixed number of generations, observing convergence in the population, or achieving a solution that meets a desired fitness level.

3.0.2. GSGP

Traditional GP methods rely heavily on the syntactic structure of programs. In classical crossover, two parent programs are combined by exchanging parts of their syntactic trees. This process can result in offspring with complex and potentially non-intuitive tree structures, which may not always align well with desired semantic outcomes. Similarly, classical mutation involves making random changes to the tree structure of a program, which can lead to inefficient or undesirable results and requires a new evaluation of the modified program.

In contrast, geometric semantic operators focus directly on the semantic content of functions. The geometric semantic crossover combines the output values of the parent functions and the random function to produce a new function, bypassing the need to handle complex syntactic structures. This method ensures that the offspring functions are created based on their ability to produce desired outputs rather than their syntactic form. Geometric semantic mutation, on the other hand, directly alters the semantic behavior of the parent function, using random functions and a mutation step to produce a new function that is semantically aligned with the intended outcomes.

In particular:

Geometric Semantic Crossover involves combining two parent functions, f_1 and f_2, with a random real function f_r whose output values lie within a specified interval $[a, b]$. Unlike classical crossover methods, which exchange syntactic parts of the parent functions, geometric semantic crossover produces a new function f_{child} by integrating the semantic information from the parent functions and the random function. The essence of this method is that it does not require retaining the tree structure of the offspring; instead, it derives the offspring directly from the semantic outputs of the parents and the random function. This approach simplifies the generation and evaluation processes, making it more efficient.

Geometric Semantic Mutation operates differently from traditional mutation meth-

ods. Given a parent function f_p, two random real functions f_{r1} and f_{r2} within the interval $[a, b]$, and a mutation step μ, geometric semantic mutation modifies the parent function by applying a semantic transformation based on these random functions and the mutation step. The resulting function f_{mutated} is produced through a process that combines the semantic characteristics of the parent and random functions, rather than altering the syntactic structure of the parent function.

3.0.3. LSGP

Local Search Genetic Programming (LSGP) is a variant of Genetic Programming (GP) that incorporates local search techniques to enhance the efficiency and effectiveness of evolutionary processes. It improves traditional GP by integrating local search strategies to refine and improve the solutions discovered by the evolutionary algorithm.

In the paper, the specific technique used is not mentioned. However, after reviewing the author's publications and other significant research on the topic, I identified two approaches that appeared both interesting and valid.

One approach to achieving this enhancement is to apply Gradient Descent within a Hybrid Geometric Semantic Genetic Programming (HYB-GSGP) framework. This method combines two steps: first, a GSGP step involving operations such as crossover or mutation, and second, a Gradient Descent step where the Adam optimizer is used to optimize the parameters of the newly generated individuals based on the semantic space.

Another approach, referred to as Half-and-Half Geometric Semantic Genetic Programming (HeH-GSGP), involves performing an equal number of GSGP steps and Adam optimizer steps. This method distinctly separates the global exploration phase from the refinement phase.

Combining these techniques ensures that the search algorithm can effectively jump to promising areas of the solution space—where high-quality solutions are likely to be found—thanks to the evolutionary search provided by GSGP and the subsequent refinement of solutions using the Adam algorithm.

4 | Data

4.1. Data sources

The dataset used in the paper study spans from January 1, 2010, to June 30, 2015, with daily observations. The key variables are summarized as follows:

- **Crude oil spot price (X0)**: The price of crude oil per barrel in USD, obtained from the U.S. Energy Information Administration.
- **EU Emission Allowances (X1)**: The price of EU Emission Allowances (EUR/tCO2), sourced from the European Energy Exchange.
- **Weather conditions (X2 to X15)**: These include air temperature, air density, wind speed, relative humidity, air pressure, sunshine duration, and precipitation (rain and snow). These were collected from the Deutscher Wetterdienst (German Weather Service)

The target variable is the **electricity price (in EUR)** for the following 24 hours.

Reconstructing the exact same dataset was impossible for me for several reasons. First, data on EU Emission Allowances is available only through payment, except for certain affiliated institutions. Second, it is impossible to retrieve data from the same meteorological station used in the paper (Frankfurt) for the same period from the Deutscher Wetterdienst website. Therefore, I chose to create a dataset that accounts for both the meteorological conditions of a region and other relevant factors, including crude oil spot prices and additional data available on energy production.

The first data source that I used relative to the energy production dataset contains four years of hourly energy production data. The dataset includes the following components:

- **Consumption and Generation Data:** Retrieved from ENTSOE, a public portal for Transmission System Operator (TSO) data.
- **Settlement Prices:** Obtained from the Spanish TSO, Red Eléctrica de España.

Column Name	Description
Time	Datetime index localized to CET.
Generation biomass	Biomass generation in MW.
Generation fossil brown coal/lignite	Coal/lignite generation in MW.
Generation fossil coal-derived gas	Coal gas generation in MW.
Generation fossil gas	Gas generation in MW.
Generation fossil hard coal	Hard coal generation in MW.
Generation fossil oil	Oil generation in MW.
Generation fossil oil shale	Shale oil generation in MW.
Generation fossil peat	Peat generation in MW.
Generation geothermal	Geothermal generation in MW.
Generation hydro pumped storage aggregated	Aggregated pumped storage hydro generation (Hydro
Generation hydro pumped storage consumption	Pumped storage hydro consumption (Hydro2) in MW
Generation hydro run-of-river and poundage	Run-of-river and poundage hydro generation (Hydro3
Generation hydro water reservoir	Water reservoir hydro generation (Hydro4) in MW.
Generation marine	Marine (sea) generation in MW.
Generation nuclear	Nuclear generation in MW.
Generation other	Other generation in MW.
Generation other renewable	Other renewable generation in MW.
Generation solar	Solar generation in MW.
Generation waste	Waste generation in MW.
Generation wind offshore	Offshore wind generation in MW.
Generation wind onshore	Onshore wind generation in MW.
Forecast solar day ahead	Forecasted solar generation.
Forecast wind offshore day ahead	Forecasted offshore wind generation.
Forecast wind onshore day ahead	Forecasted onshore wind generation.
Total load forecast	Forecasted electrical demand.
Total load actual	Actual electrical demand.
Price day ahead	Forecasted price in EUR/MWh.
Price actual	Actual price in EUR/MWh.

Forecasts and predictions have been excluded from the dataset. Only actual energy production data is retained for analysis.

The weather dataset was obtained from various weather stations. It includes detailed meteorological data collected at different locations. The dataset provides hourly mea-

surement on temperature, pressure, humidity, wind speed, and precipitation.

Column Name	Description
dt_iso	Datetime index localized to CET.
city_name	Name of the city where the data was recorded.
temp	Temperature in Kelvin (K).
temp_min	Minimum temperature in Kelvin (K).
temp_max	Maximum temperature in Kelvin (K).
pressure	Atmospheric pressure in hectopascals (hPa).
humidity	Relative humidity in percentage (%).
wind_speed	Wind speed in meters per second (m/s).
wind_deg	Wind direction in degrees.
rain_1h	Amount of rain in the last hour in millimeters (mm).

I considered only data of the Madrid Station.

Finally I also included the data for the **Europe Brent Spot Price FOB (Dollars per Barrel)** from the EIA website.

After merging all the data, I obtained hourly records from December 31, 2014, to December 31, 2018.

4.2. Data Exploration and Data cleaning

The `time` column was split into separate `date` and `time` columns, and the `time` column was further processed to extract the hour of the day as an integer. This transformation allows for a more granular analysis of the data on an hourly basis, which is crucial for understanding electricity consumption and generation patterns.

Next, the data was checked for duplicates and columns with zero or one unique value, which were then removed. This step ensures that the dataset is free from redundant or non-informative features that could negatively impact the model's performance.

Descriptive statistics were generated to understand the distribution and summary of the data, helping to identify any anomalies or patterns in the dataset. Missing values were addressed by removing columns with a high percentage of missing data, thus ensuring the integrity and quality of the dataset.

These reports can be found in the `report` folder of the project. Below are some of the available visualizations:

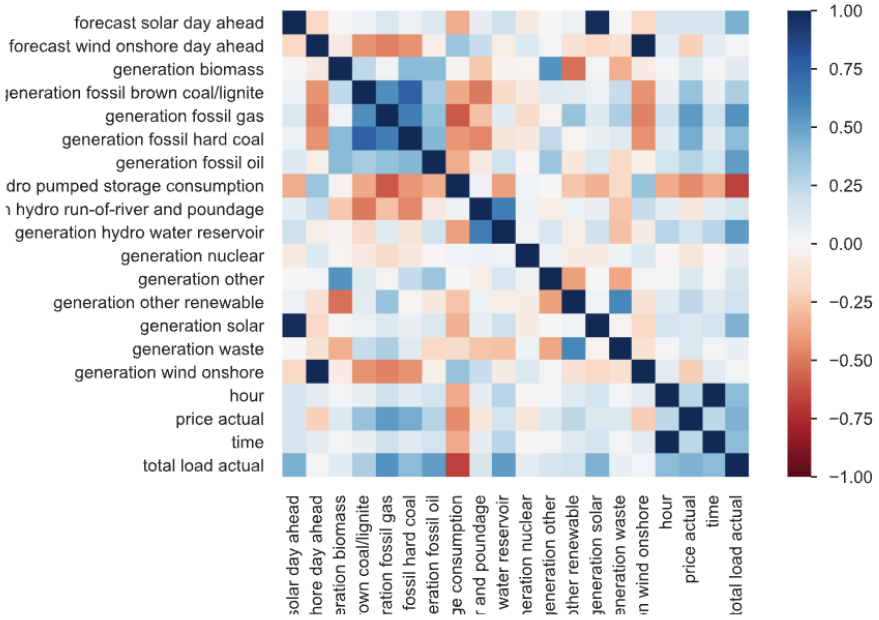

Figure 4.1: Correlations in Energy Data

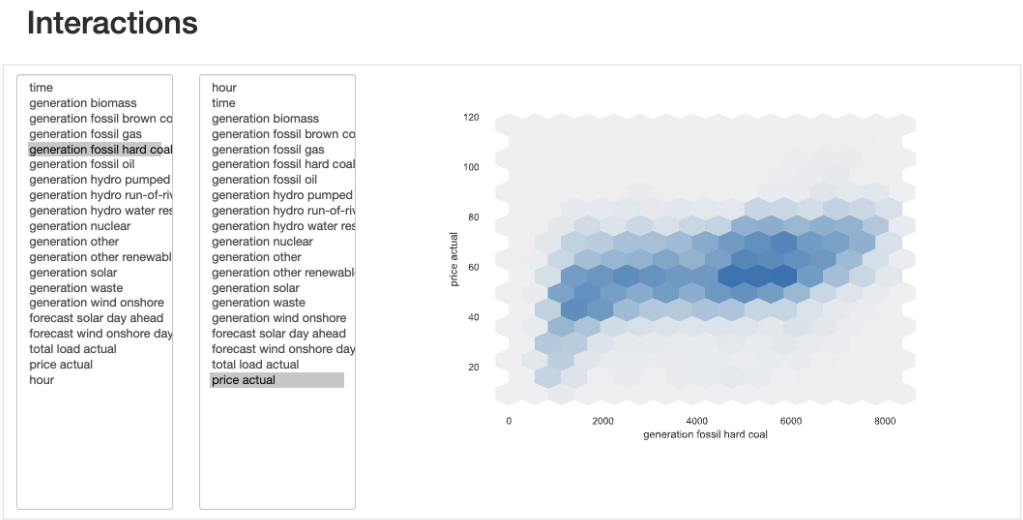

Figure 4.2: Interaction between energy price and coal production

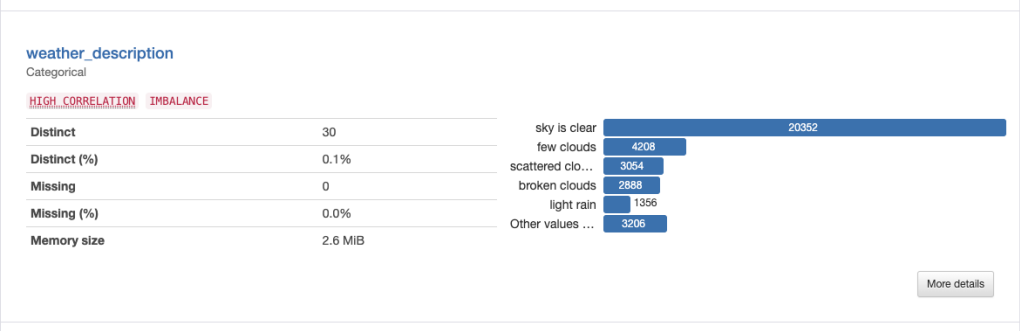

Figure 4.3: Weather descriptions distribution

Specific columns, such as `price day ahead` and `total load forecast`, were dropped as they were not needed for the subsequent analysis. One-hot encoding was applied to non-numerical data.

The cleaned dataset was then split into training, validation, and test sets. The training set (`Xtrain`, `ytrain`) is used to train the machine learning models, the validation set (`Xval`, `yval`) is used to tune hyperparameters and evaluate the model's performance during training, and the test set (`Xtest`, `ytest`) is used to assess the final model's performance. This split ensures that the model is trained and evaluated on different subsets of the data, promoting better generalization to unseen data.

5 | Experiments

The experiments conducted include the following models:

- A baseline model that predicts y_{test} as the mean of y_{train}.
- A linear regression model.
- Ridge regression.
- Random Forest.
- XGBoost.
- A standard Genetic Programming (GP) algorithm.
- An implemented Geometric Semantic Genetic Programming (GSGP) algorithm.
- An implemented Local Search Genetic Programming (LSGP) algorithm.

All algorithms, except for the last two, were fine-tuned using Optuna for parameter optimization, leveraging cloud computation (Kaggle).

I firstly attempted to implement the GSGP algorithm in C++ using the author's library directly. However, it encountered a memory issue, so I had to reimplement it entirely in Python. As a result, the implementation is computationally slow, and parameter exploration was significantly limited. Nonetheless, this process was valuable for fully understanding the algorithm's logic.

About the implementation of the LSGP algorithm, I referred to the paper "Combining Geometric Semantic GP with Gradient-Descent Optimization" and implemented both the HYB-GSGP and HeH-GSGP algorithms described earlier. In this case, computation has proven to be particularly slow, especially during the gradient calculation phase. I would like to optimize it to be able to run the algorithm on the entire dataset with parameter tuning. Currently, the algorithm operates within reasonable times only when evaluating very small populations per iteration, which makes the learning unstable. Nevertheless, the algorithm is functional, although I have not yet been able to assess its effectiveness accurately.

To see the implementation details you can see the code.

6 | Results

I will show here the performance of the various algorithms and the behaviour of the LSGP in both versions.

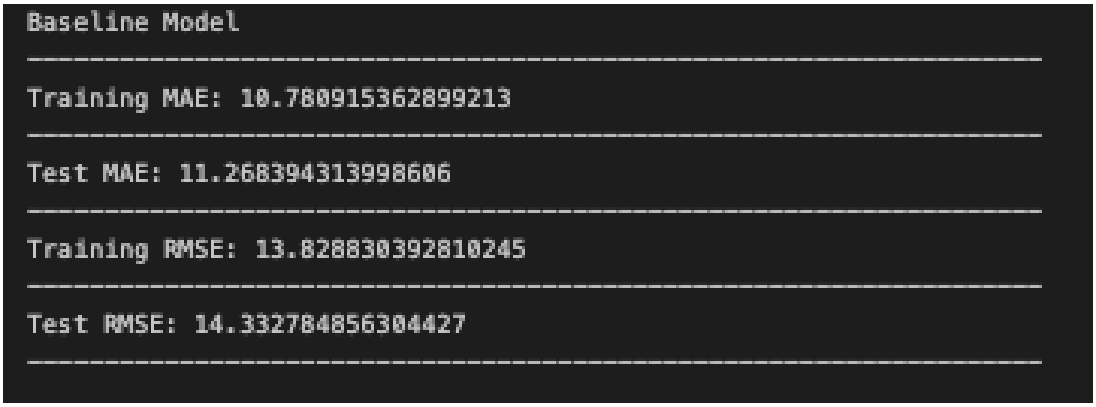

Figure 6.1: Baseline Performance

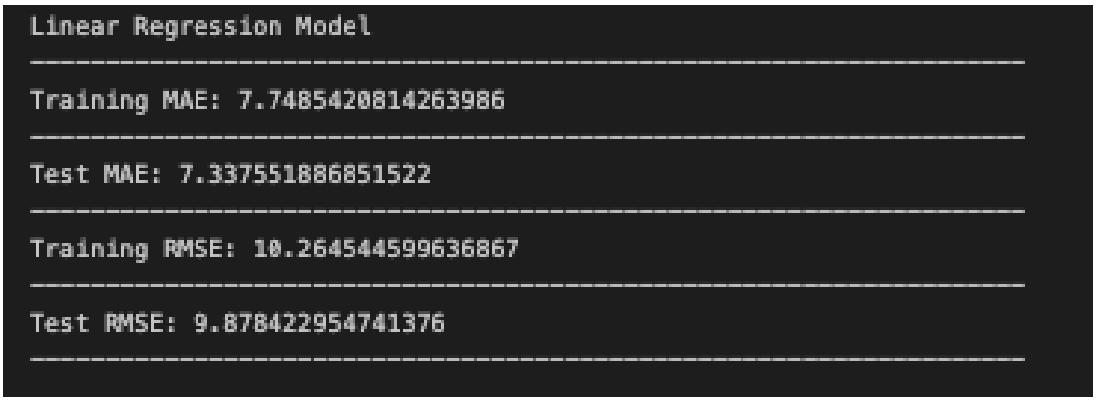

Figure 6.2: LinearRegression Performance

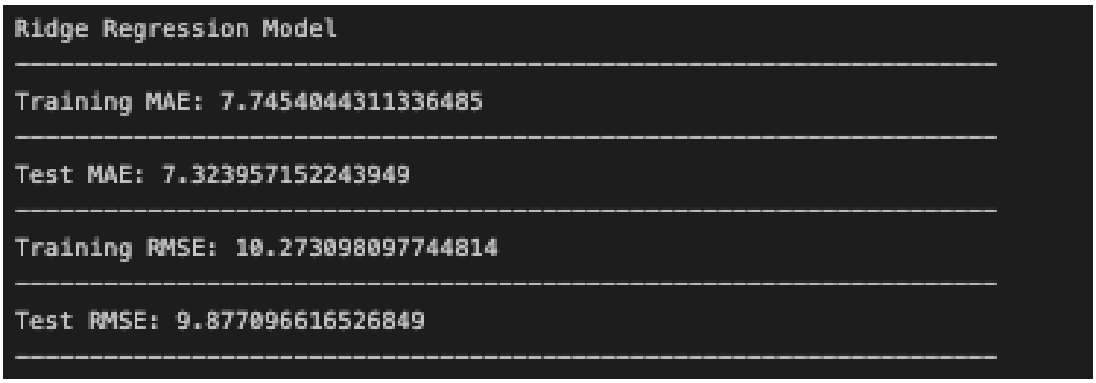

Figure 6.3: RidgeRegression Performance

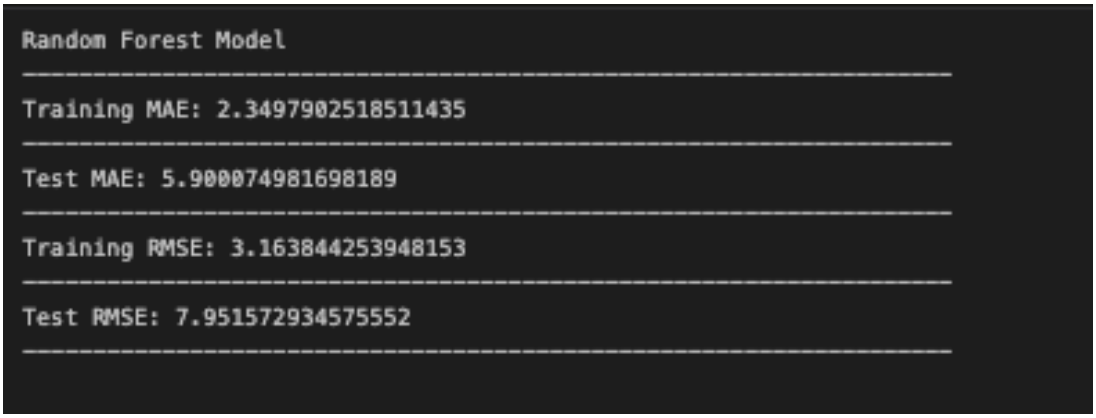

Figure 6.4: RandomForest Performance

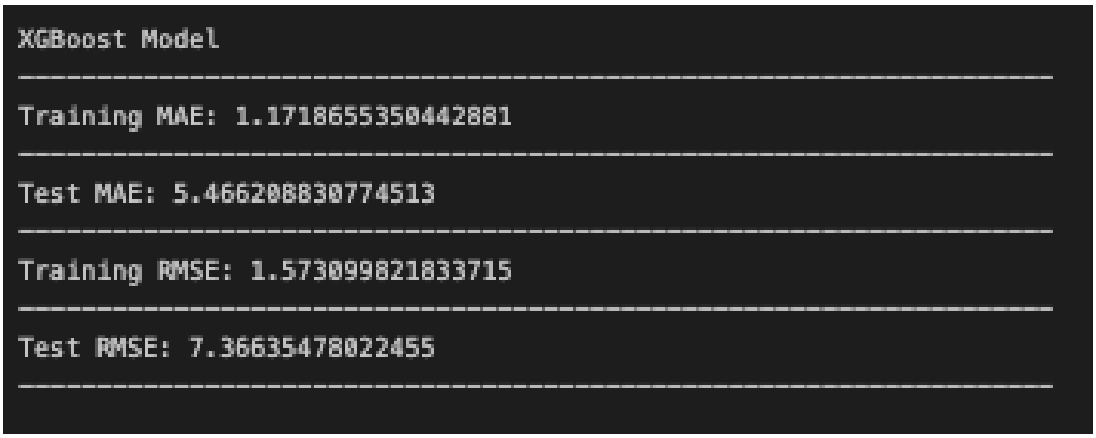

Figure 6.5: XGBoost Performance

```
Training MAE: 8.41942726989568
--------------------------------------------------------------------
Test MAE: 8.134973328640527
--------------------------------------------------------------------
Training RMSE: 11.30135896707606
--------------------------------------------------------------------
Test RMSE: 10.887546402874916
--------------------------------------------------------------------
```

Figure 6.6: GP Performance

```
Generation 0: Best Fitness = 29.517975625710612
Generation 10: Best Fitness = 15.55880441678803
Generation 20: Best Fitness = 14.167745469150073
Generation 30: Best Fitness = 14.028618807373501
Generation 40: Best Fitness = 13.312201145056635
Generation 50: Best Fitness = 13.30157987914756
Generation 60: Best Fitness = 13.285071706374017
Generation 70: Best Fitness = 13.285071706374017
Generation 80: Best Fitness = 13.278134423141138
Generation 90: Best Fitness = 13.278134423141138
Generation 100: Best Fitness = 13.278028115368357
Generation 110: Best Fitness = 13.091382575373466
Generation 120: Best Fitness = 13.082116429771396
Generation 130: Best Fitness = 13.067206292971587
Generation 140: Best Fitness = 13.056771713948432
Generation 150: Best Fitness = 12.996557790808057
Generation 160: Best Fitness = 12.996557790808057
GSGP Model
--------------------------------------------------------------------
Training MAE: 9.984149699196342
--------------------------------------------------------------------
Test MAE: 9.783147611528127
--------------------------------------------------------------------
Training RMSE: 12.996557790808057
--------------------------------------------------------------------
Test RMSE: 12.918315699527058
--------------------------------------------------------------------
```

Figure 6.7: GSGP (ON VERY SMALL POPULATION SIZE AND FEW EPOCHS)

```
HeH-GSGP Genetic step 1/8, Loss: 61.98128568227951
HeH-GSGP Genetic step 2/8, Loss: 59.00643301012296
HeH-GSGP Genetic step 3/8, Loss: 52.533631666927825
HeH-GSGP Genetic step 4/8, Loss: 49.4312377685759
HeH-GSGP Genetic step 5/8, Loss: 40.08113248493607
HeH-GSGP Genetic step 6/8, Loss: 23.069636549962468
HeH-GSGP Genetic step 7/8, Loss: 17.293726528743118
HeH-GSGP Genetic step 8/8, Loss: 17.237499474870376
HeH-GSGP Adam step 1/7, Loss: 16.31772151863793
HeH-GSGP Adam step 2/7, Loss: 15.846768516453432
HeH-GSGP Adam step 3/7, Loss: 15.376516206088406
HeH-GSGP Adam step 4/7, Loss: 14.933603107243533
HeH-GSGP Adam step 5/7, Loss: 14.534523121282932
HeH-GSGP Adam step 6/7, Loss: 14.193806396689553
HeH-GSGP Adam step 7/7, Loss: 13.916057110161882
```

Figure 6.8: LSGP-HeH

```
HYB-GSGP Generation 1/11, Loss: 62.870913507146625
HYB-GSGP Generation 2/11, Loss: 62.33294902801404
HYB-GSGP Generation 3/11, Loss: 62.43731674414295
HYB-GSGP Generation 4/11, Loss: 61.81132110524777
HYB-GSGP Generation 5/11, Loss: 61.419278308543355
HYB-GSGP Generation 6/11, Loss: 60.33971655991106
HYB-GSGP Generation 7/11, Loss: 59.76284992276528
```

Figure 6.9: LSGP-HYB

As you can see GP do not perform very well even after many hours of hyperparameters tuning. Even if GSGP performance are even worst, there is a great room of improvement in this case.

7 | Conclusion

In conclusion, I found the project both enjoyable and of personal interest. I will continue working on the implementation of the last two algorithms, particularly focusing on optimizing the parameters of the GSGP and testing the effectiveness of the LSGP in a consistent manner.