Energy Consumption Prediction via Machine Learning Algorithms

Stefanos Kontos

Supervisor: Georgios Chalkiadakis, Associate Professor ECE TUC
Vasilis Samoladas, Associate Professor ECE TUC
Fotios Kanellos, Associate Professor ECE TUC

A thesis submitted in fulfilment of
the requirements for the degree of
Diploma in Electrical and Computer Engineering

School of Electrical and Computer Engineering
Technical University of Crete

November 2021

Acknowledgments

Many people were crucial for me across this journey and I want to thank each one of them separately. First and foremost my parents, Irene and Dimitris who are standing for me and push me into testing my limits and exceeding them. Subsequently, I want to thank my girlfriend Andriana for her wholehearted support and patience all these years that kept me going.

I am also grateful to my other family, that I made in Chania. My dear friends and neighbors, George, George and Spyros whom I spent almost everyday in Chania with. Special thanks to Ilias, the fifth member of our team, that I met in the toughest period of my student life. He was the key factor of my comeback and my partner in almost every project. Also thanks to Marilena, Sophia, Ioanna, Stella and Persefoni who were always there with their positive energy and smiles.

Last but not least, I could not forget our projects' team. My mentors, Associate Professor Georgios Chalkiadakis, Dr. Charilaos Akasiadis and my teammate Stavros Orfanoudakis, for being always available and willing to help whenever I asked.

Abstract

As time goes by renewable energy usage in the residential market rises. Electric vehicles usage is in a remarkable upside and this constitutes enough excuse for researchers to invest in upgrading the electricity grid. For that purpose, the Power TAC competition provides a multi-agent simulation platform for electricity markets. In this platform, adversary brokers compete into buying and selling energy. Their primary target is to obtain max profit. The platform consists of retail, wholesale, balancing and tariff markets, which push the complexity of the broker's strategy up. Teams need to construct a broker that can manoeuvre with flexibility among consumers, producers and markets in order to accumulate profits. One such agent was TUC-TAC 2020, the agent that represented the Technical University of Crete in the PowerTAC 2020 international competition, and which was developed by a team of students in which this thesis author participated. TUC-TAC was crowned the PowerTAC 2020 champion, competing against 7 other agents representing universities from 6 different countries. In this thesis, we present TUC-TAC's energy consumption predictor module. The goal in our thesis was to predict the consumption of our agent's customers for the future

timeslots. The problem was mainly approached with Machine Learning as a regression problem. Neural Networks were also implemented and tested. The predictor module is integrated to the agent in order to provide information useful for the decision making in the tournament environment. All the different approaches are presented in detail with experimental results and comparisons. We believe that this work can serve as a departure point to build even more successful trading agents in the future.

Περίληψη

Με την πάροδο των χρόνων η χρήση ανανεώσιμων πηγών ενέργειας σε κατοικίες αυξάνεται. Επίσης η χρήση των ηλεκτρικών αυτοκινήτων είναι σε αξιοσημείωτη αύξηση και αυτό αποτελεί αιτία για επένδυση σε ερευνητικό κομμάτι ώστε να αναβαθμιστεί το ηλεκτρικό δίκτυο. Για το λόγο αυτό ο διαγωνισμός Power Trading Agent (PowerTAC) παρέχει μια πλατφόρμα προσομοίωσης πολλαπλών πρακτόρων για αγοραπωλησίες ηλεκτρικής ενέργειας. Σε αυτή την πλατφόρμα αντίπαλοι πράκτορες - μεσίτες ανταγωνίζονται στις αγοραπωλησίες. Ο κύριος σκοπός τους είναι να αποκομίσουν το μέγιστο δυνατό κέρδος. Η πλατφόρμα αποτελείται από χονδρικό και λιανικό εμπόριο καθώς επίσης και από αγορά τιμολόγησης αλλά και αγορά εξισορρόπησης τιμών, οι οποίες ανεβάζουν την πολυπλοκότητα της στρατηγικής του μεσίτη. Η κάθε ομάδα φτιάχνει το δικό της μεσίτη ο οποίος πρέπει να είναι ευέλικτος ανάμεσα στους καταναλωτές, τους παραγωγούς και τις αγορές αποσκοπόντας στο κέρδος. Για το σκοπό αυτό μια ομάδα φοιτητών ΗΜΜΥ στην οποία συμμετείχε ο συγγραφέας, δημιούργησε τον TUC-TAC 2020, τον πράκτορα που

εκπροσώπησε το Πολυτεχνείο Κρήτης στον διαγωνισμό PowerTAC 2020, ο οποίος στέφθηκε πρωταθλητής του διαγωνισμού. Σε αυτή τη διπλωματική, παρουσιάζουμε τη στρατηγική του πράκτορα από την πλευρά της πρόβλεψης κατανάλωσης ενέργειας των πελατών. Ο στόχος είναι να παράγουμε προβλέψεις της ζήτησης των πελατών του πράκτορά μας για μελλοντικές χρονικές στιγμές. Το πρόβλημα προσεγγίστηκε κυρίως με κλασικές μεθόδους Μηχανικής Μάθησης, συμπεριλαμβανομένων και των Νευρωνικών Δικτύων. Το κομμάτι των προβλέψεων ενσωματώνεται στον υπόλοιπο πράκτορα με σκοπό να του παρέχει πληροφορίες που θα βοηθήσουν στη λήψη αποφάσεων κατά τη διάρκεια του διαγωνισμού. Όλες οι διαφορετικές προσεγγίσεις περιγράφονται με λεπτομέρειες, με πειραματικά αποτελέσματα και συγκρίσεις ώστε να καταλήξουμε στην καλύτερη στρατηγική πρόβλεψης κατανάλωσης για τους επερχόμενους διαγωνισμούς.

Contents

Chapter 1

Introduction

1.1 Motivation

Data is one of the most valuable resources in our age. At the same time the data of one person or a single timestamp have no value. Different data points of different people in different situations are needed for a research to start. At the same time different data may have different value for different companies. For example a dating site's data have no value to a grocery store. On the other hand Facebook's[1] data are very useful for companies like Amazon[2] and e-Bay[3].

Secondly, even if a company has the data that fit to the company products it is not guaranteed that it can be used to create profit for the company. In contrast with most - if not all - of the resources, data flows endlessly. That makes it difficult and important to sort them out and focus on what makes impact in each field.

Once you have utilized your data you can start using it for your interest focusing on what you are learning and what is their reflection to your project. Therefore calling data the most valuable resource is kind of inaccurate but it is not wrong. Data can be gold in one man's hands and coal in another's.

[1] www.facebook.com
[2] www.amazon.com
[3] www.ebay.com

The main motivation of our research was to observe data and select what to keep and what to toss. The Smart Grid offers a pretty realistic environment with a lot of different variables that also exist in the "real world". The PowerTAC is a real time retail electricity trading competition in a simulation platform. Producers/prosumers need to process data and learn in order to strategize in Smart Grid settings, where they compete with each other. The objective of our project was to create a functional predictor that takes the data of the competition's environment as input, filters them to keep what is useful and put them into Machine Learning/Deep Learning algorithms to create predictions for the energy usage of our broker's customers for the upcoming hours or days.

1.2 Thesis Contributions

The aim of this project was to create a predictor module for the TUC-TAC agent to participate in the PowerTAC 2020 competition. For that purpose, many different approaches were implemented. The predictor was approached as a regression problem, starting with linear regression algorithm. More Machine Learning and Deep Learning algorithms were also examined. Namely Linear Regression, kernel Regression, Polynomial Regression, k-Nearest Neighbor, Regression via Feed Forward Neural Networks, Regression via Recurrent Neural Networks were all the methods we used. None of them came up with exceptional outcomes, although Regression via Feed Forward Neural Networks was the best overall.

1.3 Thesis Overview

The project described in this thesis represents the stages of the construction for the predictor part of the TUC-TAC agent, an agent that represented the Technical University of Crete in the PowerTAC 2020 international competition with this strategy. The overall strategy worked well, and TUC-TAC was crowned the PowerTAC 2020 champion, competing against 7 other agents representing universities from 6 different countries.

In Chapter 2, we present some basic background on fields important to this project, namely Data Science, Machine Learning and Deep Learning/Neural Networks. In Chapter 3 we describe the basic rules and the important component of the Power TAC simulated environment along with some of the strategies that dominated in past tournaments. In Chapter 4 we define each one of the algorithms that we used for the predictor and describe how we used them in detail, describe the architecture of the predictor, and the dataset creation process. Chapter 5 contains all the experimental results and the comparison of the different prediction methods used. Finally, in Chapter 6 we review our project focusing on what we achieved and what could be improved in the future.

Chapter 2

Theoretical Background

Predictive analytics is frequently talked about within the setting of Big Data. As data gains more value, researchers and projects aim to take advantage of this situation, analyzing and using available data to their benefit. In this chapter, terms as Data Science, Machine Learning and Deep Learning are going to be discussed and clarified. These tools are the foundations of this project.

2.1 Data Science

Data is growing in importance for any project, as it is useful for business leaders to make decisions based on statistical figures and trends. As the volume of data has increased, Data Science [22] has become a multidisciplinary entity. It extracts knowledge and insights from vast amounts of data using scientific approaches, procedures, algorithms and frameworks.

The occurring data are not guaranteed to be structured. In Data Science concepts like Data Mining, Machine Learning, and related strategies are combined to understand and analyze real-world phenomena in the form of data. Data Science is an extending Data Analysis, with areas including Data Mining, Statistics, and Predictive Analytics.

As seen in Figure 2.1, Data Science is an upcoming field that uses many concepts that belong to other disciplines such as information science, statistics, mathematics, and computer science. Some technologies used in Data Science include Machine Learning, visualization, pattern recognition, probabilistic models, data analysis, and more.

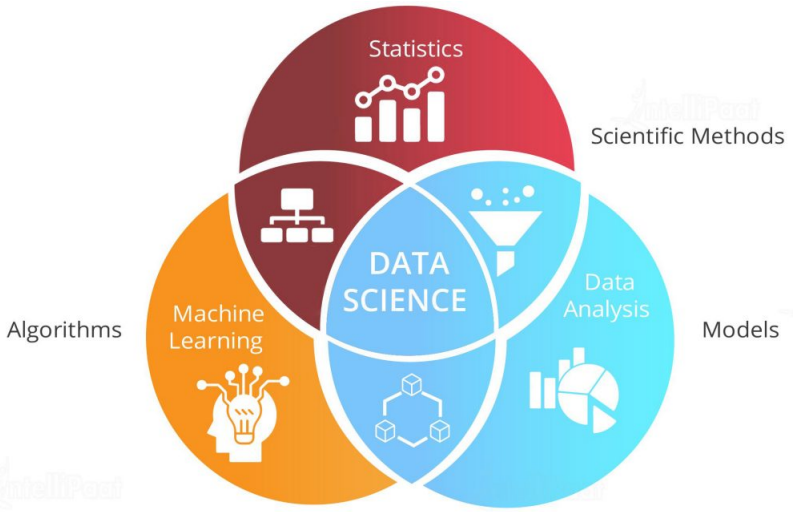


Figure 2.1. Scientific fields relevant to Data Science
https://medium.com/

2.2 Machine Learning

Machine Learning (ML) [3] uses statistics to recognize patterns in huge datasets. Data is very complicated and may include a lot of things such as numbers, characters, images and so on. The first important step is to digitally store them without losing any important information. After that, they can be processed through a Machine Learning algorithm to create a unique model. These models lie in the background of many famous services - recommendation systems like those from YouTube[1] and Spotify[2]; search engines like Google[3]; social media feeds like Facebook[4] and Twitter[5]

In any case, the platform collects as much data as it can about each individual user - what film genres they like to watch, what links they click, what status updates they respond to - and uses Machine Learning to make a thorough prediction about what the user might want next.

2.2.1 Neural Networks

Artificial Neural Networks [2] [5] are a type of Machine Learning algorithm inspired by the structure of the human brain. They may solve issues by trial and error, much like other types of Machine Learning algorithms. The output of a Neural Network is determined by the correlations between the features. Deep Learning occurs when the neurons are coupled in layers.

Nowadays, they can manage self driving vehicles, serve adverts, detect people, interpret messages, and even assist artists in the creation of new paintings. Neural Networks are used in many real-world problems today, such as speech and image recognition, spam email filtering, finance, and medical diagnosis, to name a few.

[1] youtube.com
[2] spotify.com
[3] google.com
[4] facebook.com
[5] twitter.com

Through trial and error, a learning process fine-tunes these connection strengths in order to maximize the neural network's effectiveness in solving a problem. The purpose is to find patterns and correlations between the different data features and make predictions of future values or a value for a reward function in order to solve a problem. The number and configuration of neurons in a neural network, as well as the division of labor amongst specialized sub modules, is frequently suited to each problem.

Neural Networks are only a subset of Machine Learning. A Neural Network has a big variety of use in many different Machine Learning algorithms to process complex data inputs into a space that computers can understand.

Figure 2.2 depicts a simple Neural Network. Data enters the Neural Network from the input layer. All the calculations are performed in the hidden layers, and the response is given at the output layer.

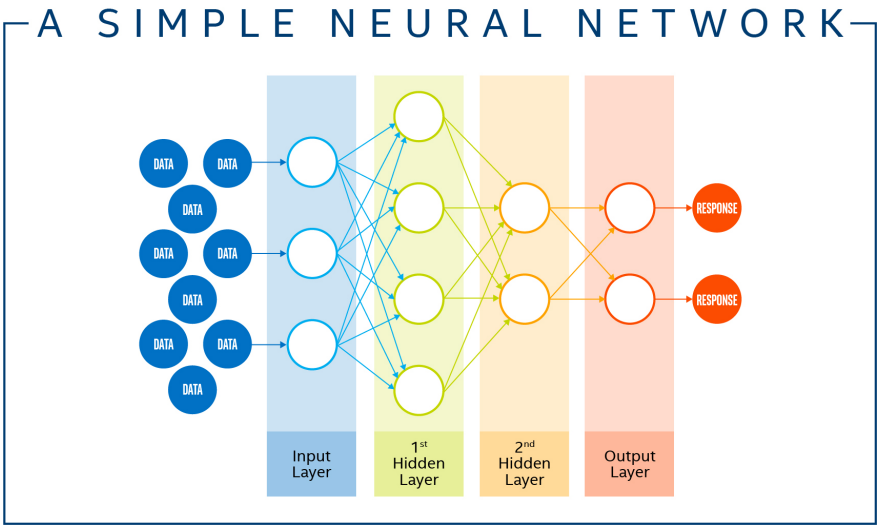


Figure 2.2. A simple Neural Network
https://simplecore.intel.com/

Chapter 3

PowerTAC: The Power Trading Agent Competition

PowerTAC is a simulated competition in a smart grid environment. In the emerging smart grid [19], automated retail brokers are expected to compete with each other on buying and selling power in both wholesale and retail markets. The critical point of each broker's strategy is achieving balance between their pricing strategies and demand, because the data is dynamic and the problem is multi parametric. Both brokers and customers are live members of this continuous price fluctuation. Wolf Ketter and John Collins created the PowerTAC simulation platform [14], that gives the opportunity to researchers to experiment with future energy market situations by creating competitive agents that face each other in yearly PowerTAC international competitions. The first PowerTAC competition was held in 2013.

3.1 Competition Overview

In PowerTAC, competitors construct autonomous trading agents that aggregate energy supply and demand targeting to earning profit. Brokers offer contracts to customers competing with each other by offering the most attractive and profitable prices and terms. This is known as the "Customers Market" part of the competition. Terms may contain fixed or varying prices for both consumption and production of energy. They may also contain bonuses or withdrawal penalties. Contract's pricing could be dynamic in order to motivate customers' investing in contracts that adjust to their needs.

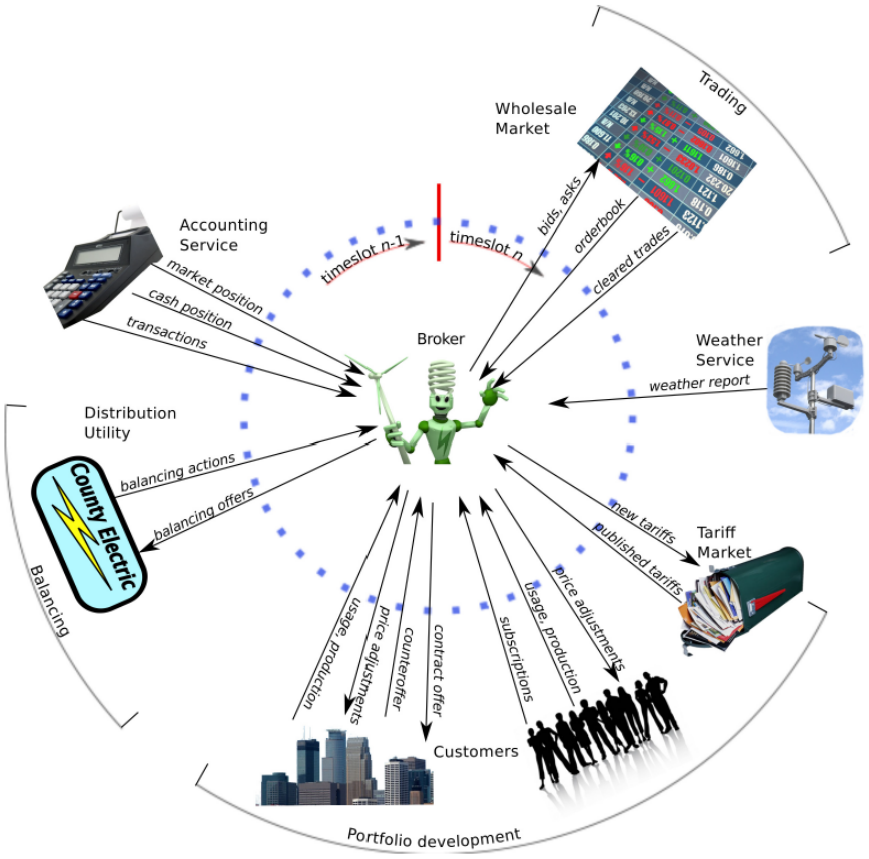


FIGURE 3.1. Available broker interactions within a time slot

As seen in Figure 3.1, Brokers need to balance their strategies in both the wholesale and the customers market. For that purpose they have a portfolio of their producers and consumers. Planning is required in order to buy the amount of energy needed to fulfil their customers needs. However, accuracy is not that easy to achieve in a dynamic and competitive market, so brokers need to delicately manoeuvre between supply and demand. The primary goal of each broker is to sign suppliers and customers to develop a qualitative portfolio. A key factor for the brokers' strategy is being able to predict the consumption wage of their customers to avoid risks of imbalance between buying and selling.

3.1.1 Simulation Time

A PowerTAC game simulation takes about two hours of time in the real world. In PowerTAC simulations, time is counted in time slots. A full game simulation runs 1440 time slots which can be converted to 60 simulated days. At any given time a time slot is the "current" time slot and there is also a set of future time slots for which the market is open for trading. The brokers have to make their moves in time to keep balance between demand and supply in each of the future time slot, otherwise monetary penalties are imposed.

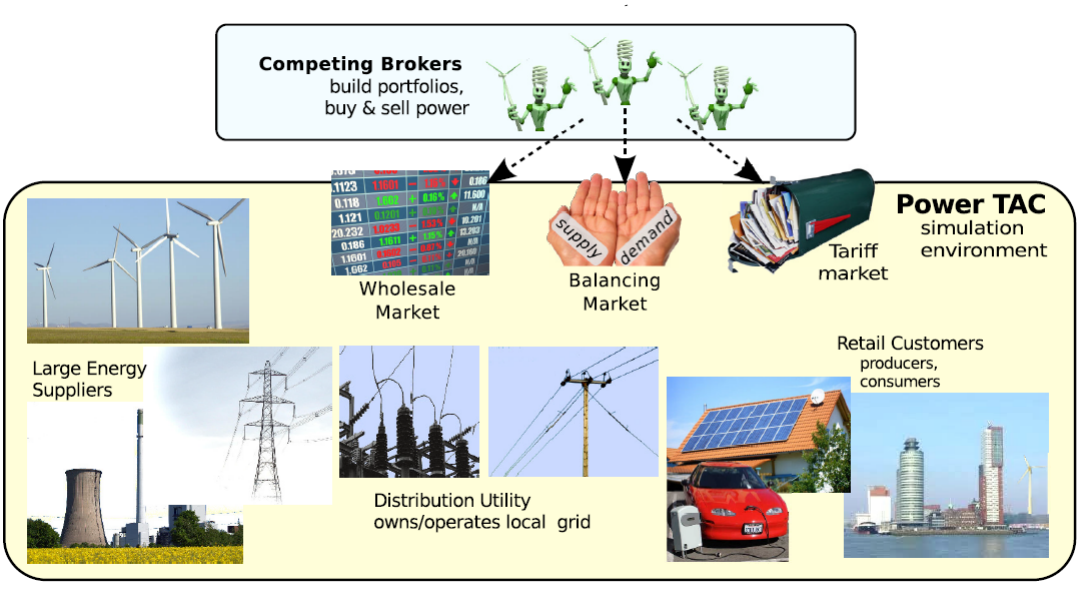


Figure 3.2. PowerTAC scenario outline

3.1.2 Brokers in the Customer Market

Brokers are the real life analogy to energy retailers. Their goal is to acquire energy from the producers and load capacity of the consumers. Brokers offer contracts with specifications like time of use, tariffs with hourly/daily intervals, sign up bonuses, withdrawal fees and dynamic prices. These are just a few of their interaction.

In more detail, brokers are able to publish new tariffs at any time or modify the existing terms by replacing their current tariff with new. In some cases, where terms allow it, price can be adjusted. Additionally, brokers can perform trades in the wholesale market by bidding for energy in the future time slots or make load curtailment to reduce overall energy in rush hours. Time also needs to be balanced and for that reason, brokers are able to submit balancing orders in balancing markets.

Focusing on the back-end of a broker, a lot of information is available for brokers to utilize, most of it coming from the bootstrap game file. Specifically, the bootstrap phase is the initialization period of the game and only the default broker is active. Its data consists of net demand and energy usage of each customer along with, data useful for the brokers to create customer models and prepare for bidding and tariff publications. Weather reports and forecasts (24 hours ahead) are also available during all phases of the game and can be useful for customer models and predictions. Brokers also have other brokers' active tariffs and bids available which are useful to make opponent models. On the other hand, transactions are private so other brokers cannot use opponents' transaction data.

3.1.3 Brokers in the Wholesale Market

In the wholesale market, brokers buy and sell energy for the upcoming time slots. The upcoming time slots are all the time slots from the next time slot up to the last of the current simulated hour when the market clears and resets. The wholesale market in PowerTAC operates as a periodic double auction (PDA) and represents a traditional energy exchange like NordPool, FERC or EEX (wholesale trading in North America and Europe).

PowerTAC cannot possibly simulate an existing wholesale market because of the geographic region covered. For that purpose, trying to make the market simulation as real as possible they created three more entities to trade along with the brokers. The first entity is "Grid Genco", representing the wide population of generating facilities that supply the simulated city. The second one is the "Grid Buyer", which simulates the regional demand based on real life metrics. The third and last entity is providing liquidity to the market by mimicking a population of buyers and speculators only in the wholesale market.

3.1.4 Balancing Market

The balancing market mimics the operation of Independent System Operator (ISO) in the United States and Transmission Systems Operator (TSO) in Europe. Note that PowerTAC does not model the full hierarchy of the organizations mentioned above. In more detail the balancing market is responsible for real time balance of supply and demand. For that purpose it tries to motivate the brokers to balance their portfolios instead of relying on the balancing market to do it for them. Whenever a broker exceeds the balance limits, the balancing market charges it for the missing energy at an unreasonably high price acting as a penalty.

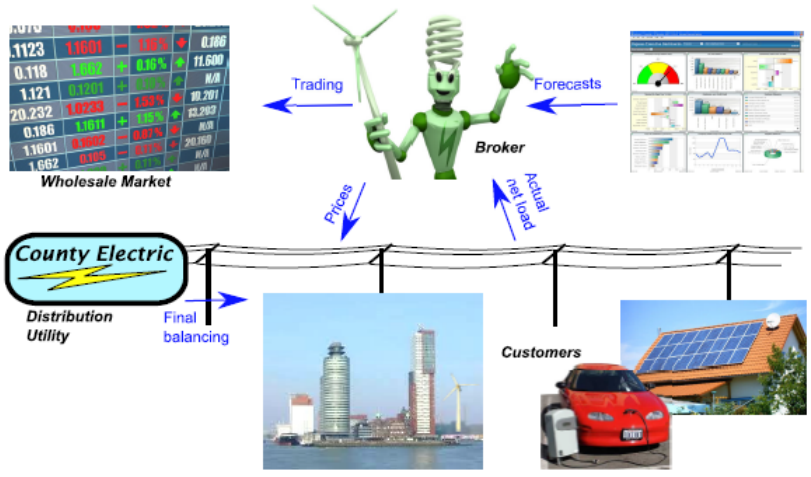


Figure 3.3. PowerTAC scenario outline

3.1.5 Customer Models

Customers interact with the brokers in the tariff market. They differ with each other in some specific characteristics according to their customer model. Each customer model is specified with some information. These characteristics differ from one model to another. In more detail, customer models differ in population size, for example there are models that refer to households and models that refer to offices. They also may differ in the power type they use, which indicates if a customer is producing, consuming or doing both. Moreover, not all customer models are able to negotiate for tariff prices or sign contracts with multiple brokers. These characteristics constitute the core set of information for each customer model and are communicated to the brokers at the beginning of each simulation.

3.1.6 Weather Reports

Each time slot agents receive weather reports and forecasts. The weather variables are temperature, wind speed, wind direction and cloud coverage. Each forecast contains the variables mentioned before for each upcoming hour for the next 24 hours in advance. Given this data, brokers are able to create prediction models to estimate the usage of their customers in the upcoming time slots. Weather data is based on real world location in the past but agents are not aware of this information until the game is over.

3.1.7 Distribution Utility

The Distribution Utility is responsible for issuing the transmission capacity fees. Whenever a demand peak arises, each broker has to pay the amount of the total fee his customers are responsible for. Distribution Utility is also responsible for issuing distribution fees to brokers, for distributing energy to customers. Therefore it is very important for the agent to buy no more energy than the customers demand in order to avoid paying fees for distribution or exceeding limits. The Distribution Utility also publishes the tariffs when brokers have no deals made.

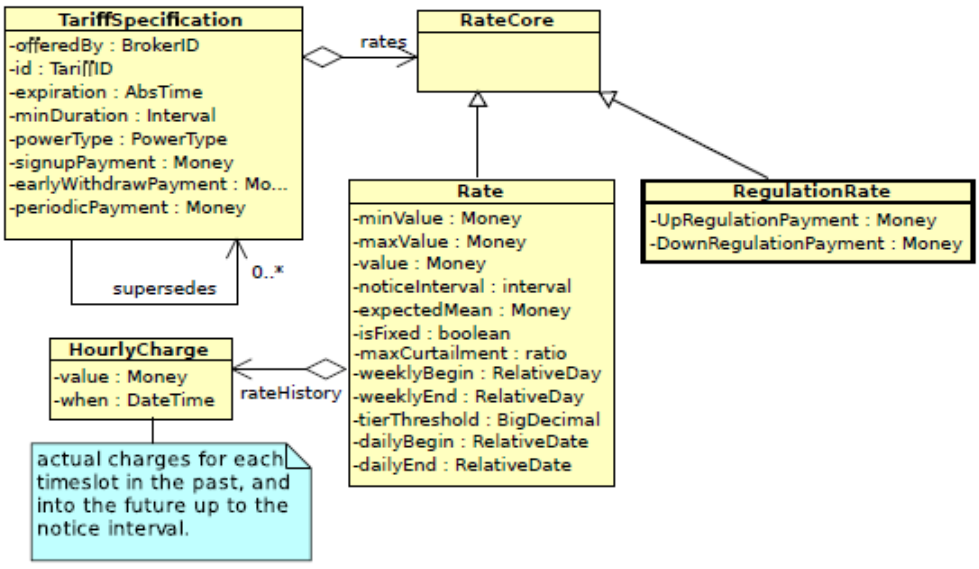


Figure 3.4. PowerTAC Tariff Specifications

3.2 Related Work

As mentioned before, this thesis project focuses in the predictor module of the PowerTAC agent. In 10 years of PowerTAC competitions many ideas were implemented, but some of them were successful enough to be mentioned. The predictor module is mainly useful in the tariff market where customers interact with the brokers in order to sign the contract that gives them the most benefit.

3.2.1 Neural Networks

In order to predict the usage of each customer, agent VidyutVanika [7] implemented a Neural Network (NN) with 2 hidden layers of size 7 each and 10 epochs of training over the training data. The features of the data set were time of day, day of week and the weather forecast variables. Each customer type has a different model that is updated through the game as the broker gets more data. The agent finished in the second place in the competition of 2018.

3.2.2 Regression Methods

Agent UDE [16] was consistently winning a place in the podium from 2016 to 2019 with 2 first places. Their prediction strategies were based on linear regression. Specifically, Agent UDE17 focuses on features like the customer type and capabilities to predict future demands. Apart from that, a seasonal autoregressive integrated moving average (SARIMA) is computed to find any periodic behavior in the energy demand of the customers. It is worth mentioning that Agent UDE predicts demand without taking into account weather conditions, and focuses on past market data.

3.2.3 Classification Methods

Mertacor2019 was the winner of the PowerTAC 2019 and the same team created the agent that was the runner-up of the PowerTAC 2020 competition. Although the agent is consistently one of the top in the tournament, the team is not focusing on the consumption predictor module. Their prediction strategy in 2019 when Mertacor was crowned winner was a simple classification method, used in order to approximately calculate customers' consumption.

3.2.4 Markov Decision Process

The Crocodile agent [10] is another consistently reliable agent. In the customer demand prediction this agent implemented a Markov Decision Process (MDP) [18] for the wholesale market and Reinforcement Learning (RL) for the retail market. This selection was previously presented by many other agents too but the Crocodile agent was mentioned because of its total good results.

Chapter 4

Implementation

The target of this project was to create a predictor module for the TUC-TAC agent to participate in the PowerTAC 2020 competition. For that purpose, many different approaches were implemented. The predictor was approached as a regression problem, starting with linear regression algorithm. Machine Learning and Deep Learning algorithms were also examined. In this chapter we will start by first describing the dataset and its features. Moreover, the architecture of our predictor and all the methods used are going to be explained in detail, starting with the underlying theory and moving on to the application of the algorithms and our experimental results.

4.1 Dataset Construction

The competition is taking place in the virtual environment of the smart grid. Despite this, the instances and the data that the smart grid contains seem pretty real. In more detail, the simulation consists of different customer models which differ in some important factors. First of all the customers are divided into three main categories according to the power type. These categories are consumers, who only buy energy from our broker, producers, who only sell energy to our broker, and prosumers that are a hybrid of the first two.

Additionally, customer models vary in population and capacity. Apart from the customer model information, the simulation provides us with the energy usage of each customer per time slot. Moreover, weather forecast and report are provided per time slot and we also keep this data in track because we consider them valuable features for the model construction. The features that are used in all the datasets for regression algorithms are:

- hour
- day
- month
- year
- temperature
- wind speed
- wind direction
- cloud cover

4.1.1 Bootstrap Phase

After referring to all the sources of data available, it is time to start selecting our dataset's features. For each one of the simulations our broker has access to one file before the game containing the bootstrap phase data. The bootstrap phase of the game is the initial phase where the game has not actually started, meaning that no negotiations have been made yet, but the environment has been set-up. Its name has the form finals2020** **.xml, where the stars are representing the unique game number that works like a unique identity for the game. This file contains 336 timeslots and the features that we are monitoring are:

- timeslot
- temperature
- wind speed
- wind direction
- cloud cover
- net usage (the total net usage of all the entities in the smart grid)

Net usage is actually our target variable and we use any past values of the available variables to help the model make better predictions on the upcoming values. Up to this point we have a dataset of features with shape of (336,5) and our target variable with shape (336,1). This is all the available data we have if we want to make online predictions. This is one of the cases we are going to discuss in this chapter but for now lets move on to collecting and explaining all the data of a simulation.

4.1.2 Post Game Collection

At the end of each game two more files are generated. They have the same name format as the bootstrap file with a different file type. These files give us some extra information. We can learn the location where the weather data were taken from, along with the full time and date of the first timeslot of the game. Weather forecasts and predictions for all the timeslots of the game are also available. We collect all this data aiming to find possible correlations between the games that took place in the same location or at the same dates.

After collecting all the data mentioned above for all the games of the 2020 finals, we divided the data by location. We have three different locations, Denver Phoenix and Minneapolis. The total number of datasets that we created is seven. There is one dataset that contains all the collected data irrespectively of location. Three more datasets contain one location each, and the last three contain all the possible couples of locations.

Concluding, the features in all the mentioned data are the same eight:

- hour
- day
- month
- year
- temperature
- wind speed
- wind direction
- cloud cover

The number of datapoints differs from one dataset to another, specifically the data shapes are:

- Denver(88604,8)
- Minneapolis(89376,8)
- Phoenix(79038,8)
- Denver & Minneapolis(177979,8)
- Denver & Phoenix(167641,8)
- Minneapolis & Phoenix(168413,8)
- Denver & Minneapolis & Phoenix(257016,8)

4.1.3 Data Correlation

The next step was to examine the correlations of each feature with the target value. Unfortunately, according to the scatter plot of Figure 4.1 none of the features mentioned was related enough with the net usage value. This is visible in the last row/column of the scatter plot below with the label "mwh". For that purpose, a lag feature was added to our datasets. A lag feature is a feature that in timeslot t takes the value of the target value in timeslot $t-1$. Adding a second lag feature extends the lag giving the value of the target in $t-2$ and so on.

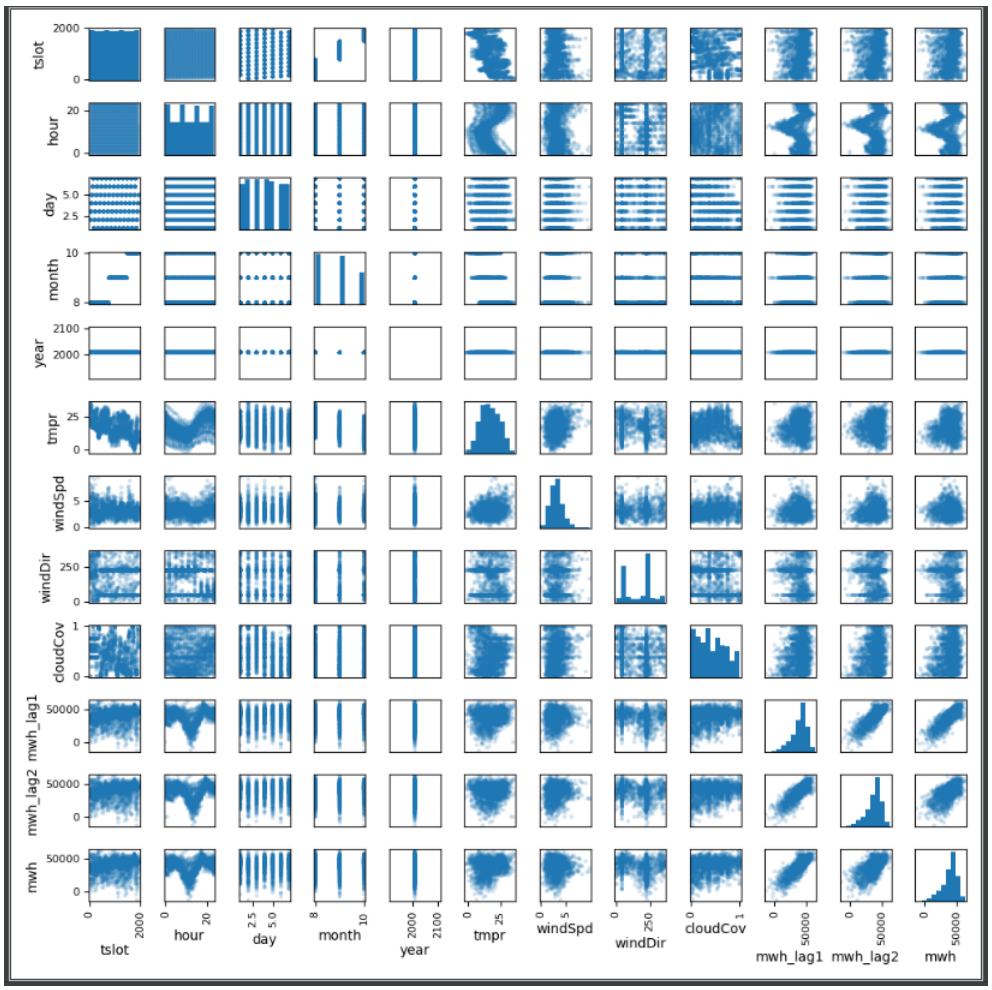


Figure 4.1. Scatterplot with all the features of the dataset

4.2 Predictor's Architecture

In order to maintain fast and reliable connection with the main agent we set up a python server. In this communication, all the useful data are sent to the predictor via json files and predictions are returned. The main functions of the predictor are demonstrated in the diagram below and described in the next subsection.

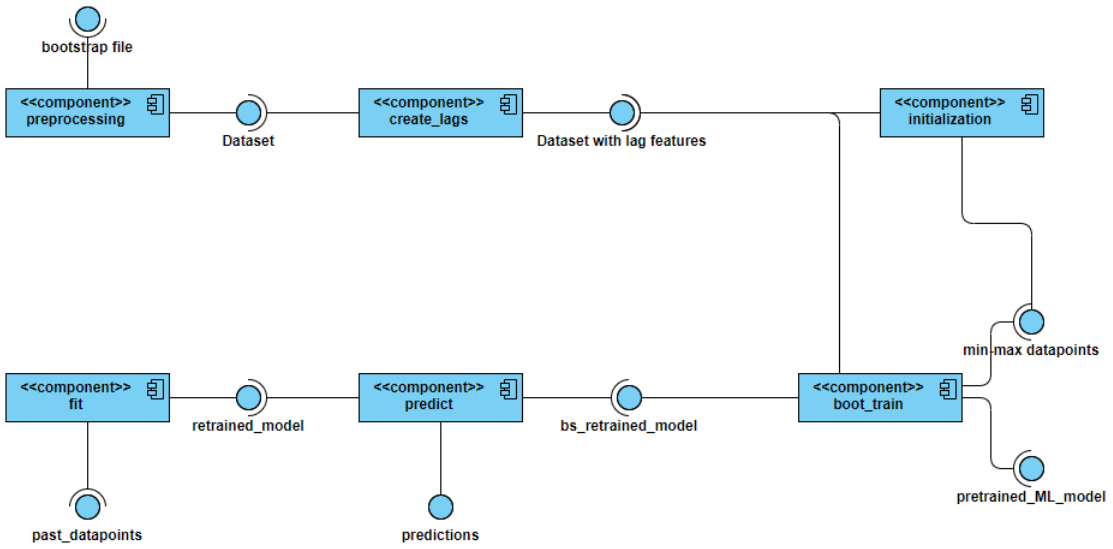


FIGURE 4.2. Predictor's architecture

4.2.1 Main Algorithmic Components

In this section, the main components of the predictor, with their inputs and outputs, are going to be described in detail. Regardless the algorithm used for the predictor the compoments used are the same.

4.2.1.1 Preprocessing

The first stage is to take the bootstrap file, or any log files, in text form, extract the useful information and create a dataset.

4.2.1.2 Create Lags

This component takes the dataset and the number of lag features as input and returns the dataset with the lag features added.

```
def create_lags(dataset, num):
    new_dict = {}
    for col_name in dataset:
        new_dict[col_name] = dataset[col_name]
        # create lagged Series
        for l in range(1, int(num) + 1):
            new_dict['%s_lag%d' % (col_name, l)] = dataset[col_name].shift(l)
    res = pd.DataFrame(new_dict, index=dataset.index)
    res = res.fillna(0)
    return res
```

Figure 4.3. Python Code for creating lag features

4.2.1.3 Initialization

This component loads a big dataset and constructs 2 new "fake" datapoints with the minimum and maximum values for each feature. These datapoints are not actual datapoints from past games. They are custom with purpose to set the minimum and maximum values of each feature. That is really important because datapoint values are scaled in the (0,1) before getting fit into the model. In order for the scale to work properly same upper and lower limits are needed in every case.

4.2.1.4 Boot Training

This function is used only in the bootstrap phase of the game. We take bootstrap dataset, load our pre-trained model, fit the bootstrap data in it and then save the updated one.

4.2.1.5 Fit

This component is used when past data becomes available. In this function past datapoints are given as input and the predictor fits them into the model to gain more "knowledge". This was decided because of the deviation between the predictions and the real values. Actually this is the module where the model is getting retrained and creates new weights because of the new values that are fitted in it. The output is a new retrained model after some online training.

4.2.1.6 Predict

In this function the input is the features of the datapoints we want predictions for. The predictions are created according to the model's weights and sent back to the broker. However they are not fitted to the predictor model. In case we fit the predicted values into the model, upcoming prediction are going to have even higher deviation from the real values because the errors are going to be fitted as "correct" values. In more detail this is the component that makes the prediction based on the current predictor model. Specifically, the models we created that are going to be described in the next section are:

- Feed Forward Neural Networks for Regression
- Recurrent Neural Networks for Regression
- Linear Regression
- Kernel Regression
- k-Nearest Neighbor

4.3 Artificial Neural Networks

Neural Networks are beneficial for situations with complex or poorly understood solutions. They are suitable for solving problems with big amounts of data containing multiple variables that are beyond human grasp. Neural Networks are useful for pattern recognition, which means matching the inputs according to their statistical variation.

Our problem is one with high complexity given the many different variables that have influence on the net usage value. In our situation we are implementing two different types of Neural Networks, Feed Forward Neural Networks and Recurrent Neural Networks. Feed Forward and Recurrent Neural Networks are going to be used for Regression.

4.3.1 Feed Forward Neural Networks (FFNNs)

The fundamental Deep Learning models are FFNNs, known as multilayer perceptrons (MLPs). FFNNs' purpose is to approximate a function f*. In our project FFNNs were implemented for both Regression and Classification. The architecture of our Neural Network is very similar to the Neural Network in Figure 4.4, the only difference is the number of neurons. The input layer contains the features of our dataset, each feature is represented from one neuron.

In our Regression case the features are 9, and thus we have 9 input layer neurons. Then we have 2 hidden layers as presented in the Figure, but the size of each is 24. In these layers the Neural Network makes calculations and decides the weights of the edges. These weights represent the correlations between the features and target value. The last layer is the output layer in which the result is posted. The output layer contains only one neuron.

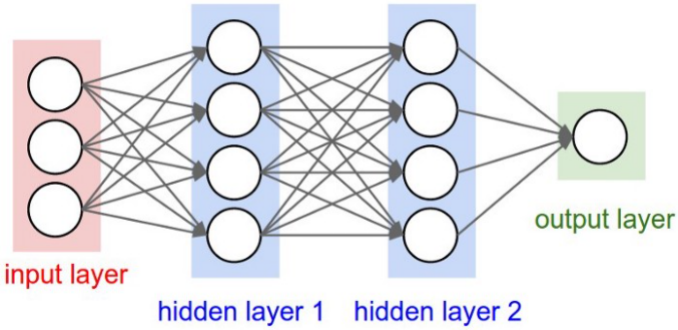


Figure 4.4. A simple FFNN Architecture with 2 hidden layers

4.3.1.1 Feed Forward Neural Networks for Regression per Customer

The idea of having one FFNN for regression per customer, was one demonstrated in the VidyutVanika agent [7] in 2019 Power TAC tournament. In more detail, a unique model was trained on the NN for every customer type. At the end of each simulated day a prediction was calculated for each customer type and all the predictions we aggregated to summarise the total net usage of our customers. Theoretically this approach could help limiting our miscalculations, because of error mutual neutralization.

4.3.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are based on Feed Forward Neural Networks with an extension of memory (feedback). For that reason, RNNs are suitable for time series or sequential data like Natural Language Processing (NLP). Their feedback asset gives the opportunity to alter the input of the Neural Network by using past inputs. While typical deep neural networks believe that inputs and outputs are independent of one another, recurrent neural networks' output is reliant on the sequence's prior components.

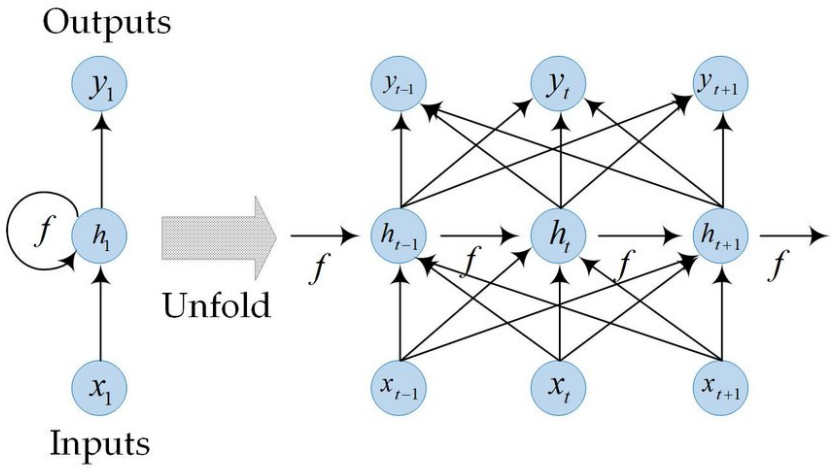


Figure 4.5. RNN unfolded
https://www.researchgate.net/figure/Recurrent-neural-networks-structure

In this figure we can see a simple architecture of a Recurrent Neural Network. On the left, the figure represent the whole Neural Network with the input, the hidden layer and the output. On the right side, the Neural Network is presented unfolded, showing all the time steps.

4.4 Linear Regression

Linear Regression is a predictive analytic technique that uses a statistical linear machine learning algorithm. Linear regression methods demonstrate a linear connection between a dependent variable, y, and one or more independent variables, x, i.e., how the value of the dependent variable, y, varies as the independent variable's value changes. Simplified, Linear Regression's purpose is to create a projection that is as close as possible to all the datapoints of the dataset.

For the implementation of linear regression we used the scikit-learn library [17]. For the rest of the work we did pretty much the same things. We created the lag features, observed the same metrics as in the regression with the NNs and split the datasets with the same 90-10 principle.

```
model = LinearRegression()
model.fit(x_train, y_train)
model = LinearRegression().fit(x_train, y_train)
y_hat = model.predict(x_test)
y_hat = scl.inverse_transform(y_hat)
```

Figure 4.6. Fitting data into the model and predicting values

4.5 Kernel Regression

Kernel Regression [1] is a data-fitting estimating approach. One wishes to discover a regression function $f_a(x, w)$ that is the best-fit match to the available data at those data points provided a dataset (X_i, Y_j). One could also wish to extrapolate and approximate beyond available data. Unlike linear or polynomial regression, where the underlying assumption (e.g. normal distribution) is known, kernel regression does not make any assumptions about the underlying distribution while estimating the regression function. Kernel regression is classified as a non-parametric method because of this.

Kernel regression works by assigning each observational data point to a collection of identical weighted functions known as Kernel local. The kernel will give each location a weight based on its distance from the data point. The radius or width (or variance) from a local data point X to a collection of nearby locations x is all that matters to the kernel basis function. Kernel regression is a superset of local weighted regression and closely related to Moving Average and k nearest neighbor (kNN).

```
def gaussian_kernel(self, z):
    return (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * z ** 2)

def predict(self, h):
    kernels = np.array([self.gaussian_kernel((np.linalg.norm(xi - h)) / self.b) for xi in self.x])
    weights = np.array([len(self.x) * (kernel / np.sum(kernels)) for kernel in kernels])
    return np.dot(weights.T, self.y) / len(self.x)
```

Figure 4.7. Implementing the Gaussian Kernel and make a prediction

4.6 k Nearest Neighbors (kNN)

The supervised machine learning method k-nearest neighbors (kNN) [1] may be used to tackle both classification and regression problems. kNN, is arguably a "real-world" algorithm: People are influenced by those around them. Our behavior is influenced by the companions with whom we grew up. In some respects, our parents influence our personalities as well. If you grow up with people who love sports, it is highly likely that you will end up loving sports. Of course, there are exceptions. kNN works similarly. The majority voting principle is used by the kNN classifier to identify the class of a data item. When the value of k is set to 5, the classes of the five nearest points are examined. The majority class is used to make predictions. Similarly, the mean value of the five nearest points is used in kNN regression.

kNN is an algorithm that works in both classification and regression tasks. It can work really well in multiple class classification. It can also be implemented in non-linear tasks as it makes no assumptions. On the other hand kNN becomes very slow as the number of data points increases because the model needs to store all data points. This means it is not memory efficient.

4.7 A Heuristic approach

In this part of the project we tried to approach the problem by observing the data as if Smart Grid was a reality. The idea was to check if real world assumptions exist in the Smart Grid environment in order to set stricter limits to our predictor. We wanted to check for habits that could lead into repetition in the usage. Different observations were made from different perspectives.

At first, we compared the same groups of hours of different days, then days of weeks, days of months, workdays, work hours, sleeping hours, weekends, first days of the month last days of the month. Another perspective was to observe each one of the three locations on its own to find some special features.

This heuristic search for patterns did not result in any major improvements in our results, though we can certainly infer that "life" in the Smart Grid does follow some of the same habits with the real world, for example the peak usage is in the evening and the lowest usage values in the morning.

Chapter 5

Experimental Results

In order to compare and contrast the results of each algorithm we had to evaluate them given the same terms. The same datasets were used for all the algorithms and the same accuracy metrics were scoped. Apart from the accuracy metrics, execution time is also important in order for the module to be useful in the competition. All the experiment result are going to be presented and explained by the type of the algorithm used. After discussing the results of each algorithm, the next step is to compare and contrast the results to each other, in order to end up with the algorithms that were more appropriate and fit better to our kind of problem. The 3 main datasets used in the experiments are Denver, Minneapolis and Phoenix, categorized by location. The fourth is a combination of the 3 above. The dataset creation is explained in detail in paragraph 4.1.2

5.1 Monitored Metrics

In each one of the 6 regression algorithms implemented we are keeping track of some metrics in order to compare and contrast the algorithms. Specifically we are measuring:

- Mean Absolute Percentage Error (MAPE): This value expresses the mean absolute distance between the real target values and the predicted values. Smaller error values mean bigger success in the algorithm predictions.

$$MAPE = \frac{1}{n}\sum \left| \frac{x_i - y_i}{x_i} \right|$$

Where y_i is the prediction and x_i is the true value.

Rooted Mean Squared Error (RMSE): This error metric is very similar to MAPE but it is not the same. RMSE is a quadratic scoring metric that tends to weigh bigger errors more than smaller ones. That is not useful in all cases but in our approach is very important because we are trying to avoid big divergence from the real values.

$$RMSE(\hat{\theta}) = \sqrt{MSE(\hat{\theta})} = \sqrt{E((\hat{\theta} - \theta)^2)} \tag{5.2}$$

For an ubiased estimator the RMSE is the square root of the variance.

R^2(R Squared): This metric is used in statistical models to measure the coefficient of determination. In other words it is a metric that monitors the correlation between the independent variables (features) and the dependent variable (target value). The best score for this variable is between 0 and 1.

If $\overline{y}$ is the mean of the observed data:

$$\overline{y} = \frac{1}{n} \sum y_i \tag{5.3}$$

The sum of squares of residuals is:

$$SS_{res} = \sum (y_i - f_i)^2 = \sum e_i^2 \tag{5.4}$$

The total sum of squares proportionally to the variance of data is:

$$SS_{tot} = \sum (y_i - \overline{y})^2 \tag{5.5}$$

Generally, the definition of the coefficient of determination is:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{5.6}$$

Execution Time in Seconds: It is very important for our strategy to use an algorithm that executes as fast as possible in order to make our prediction within the given time limit. In the Power TAC 2020 Tournament that was 10 seconds. The predictor has to give the result in 5 seconds maximum to be sure that the main agent will receive it

and take it in consideration.

5.2 Linear Regression

The experiments contain data from the PowerTAC 2020 final. Data from 145 games was merged into one big dataset containing about 257.000 datapoints. This data was then divided into 3 smaller datasets based on the location of the game. The size of the 3 datasets is similar and they contain 80.000-90.000 datapoints each. It is important to mention that according to PowerTAC rules the agent has no location knowledge before the end of the simulation.

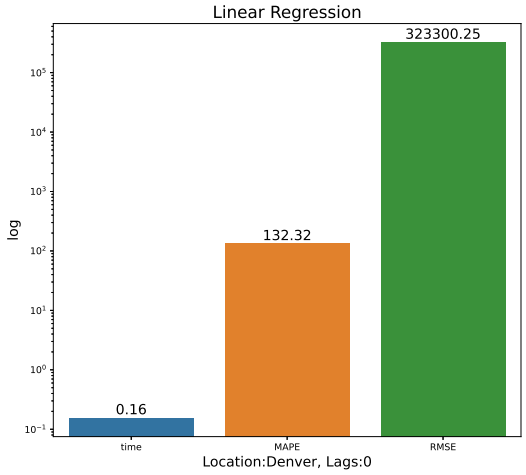
The following tables present Linear Regression results with each one of the 4 datasets, 3 small datasets based on location and 1 big dataset containing all the data combined. The number of lag features is also examined and for that purpose 0,1,2 and 4 lag features are tested for each dataset. The metrics that evaluate the performance of each test case are time, MAPE, RMSE and R^2. They were explained in detail in the previous section.

Linear Regression - Denver Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.1555933952	0.1535613537	0.1645598412	0.1866192818
MAPE	132.3196634	87.37115316	107.8967211	88.32220319
RMSE	323300.2508	2464983.433	2841246.4	2469051.99
R^2	$-1.2 * 10^{16}$	$-8.76 * 10^{15}$	$-9.04 * 10^{15}$	$-1.03 * 10^{16}$

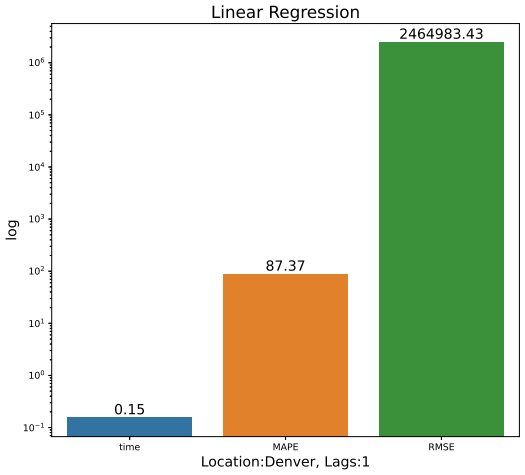
Linear Regression - Phoenix Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.1206541061	0.1326992512	0.1416511536	0.1670753956
MAPE	88.2980799	88.19853496	88.43660946	88.05734155
RMSE	38553.19332	40005.86636	40069.08924	40014.20918
R^2	$-10 * 10^{10}$	$-11 * 10^{10}$	$-11.1 * 10^{10}$	$-11 * 10^{10}$

Linear Regression - Minneapolis Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.13935256	0.1790144444	0.181383152	0.1845057011
MAPE	113.1542341	113.0788475	113.0135115	113.2855544
RMSE	43758.80845	44893.71847	44875.036	44976.67736
R^2	$-16.4 * 10^{10}$	$-17.1 * 10^{10}$	$-17.2 * 10^{10}$	$-17.1 * 10^{10}$

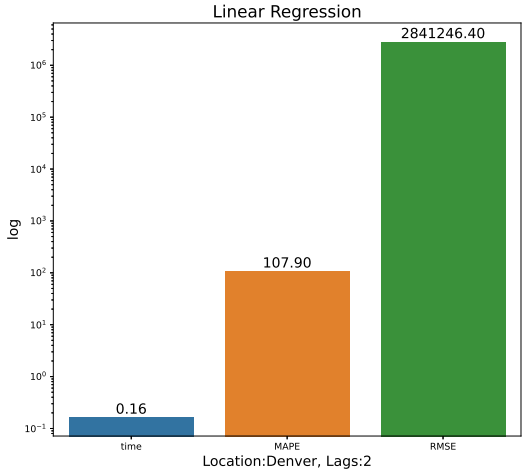
Linear Regression - Combined Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.3410875797	0.4218530655	0.4378292561	0.5106351376
MAPE	115.922184	79.63144835	113.7340657	90.55751338
RMSE	137017.8616	1200225.287	1710336.722	1378274.8
R^2	$-89.4 * 10^{12}$	$-9.76 * 10^{15}$	$-9.24 * 10^{15}$	$-8.76 * 10^{15}$



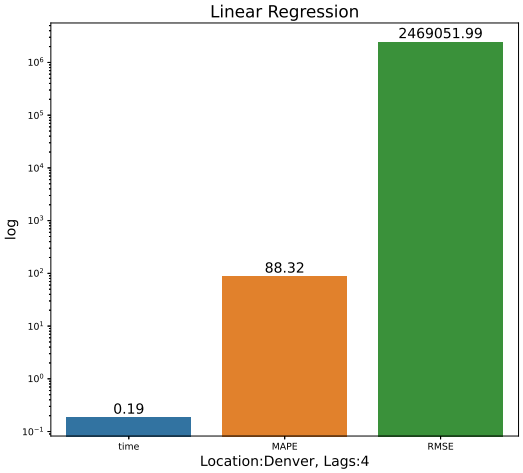
(A)



(B)

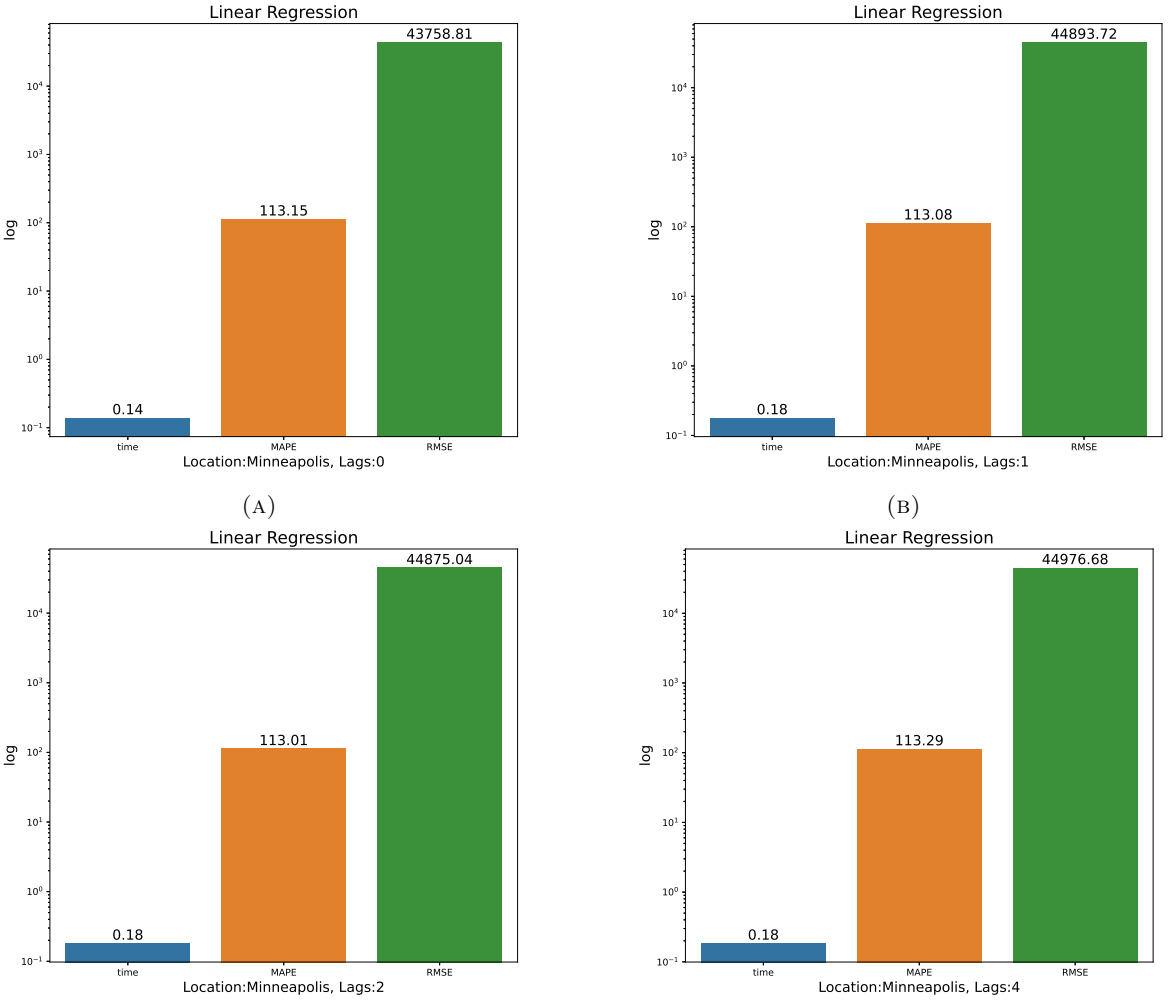


(C)



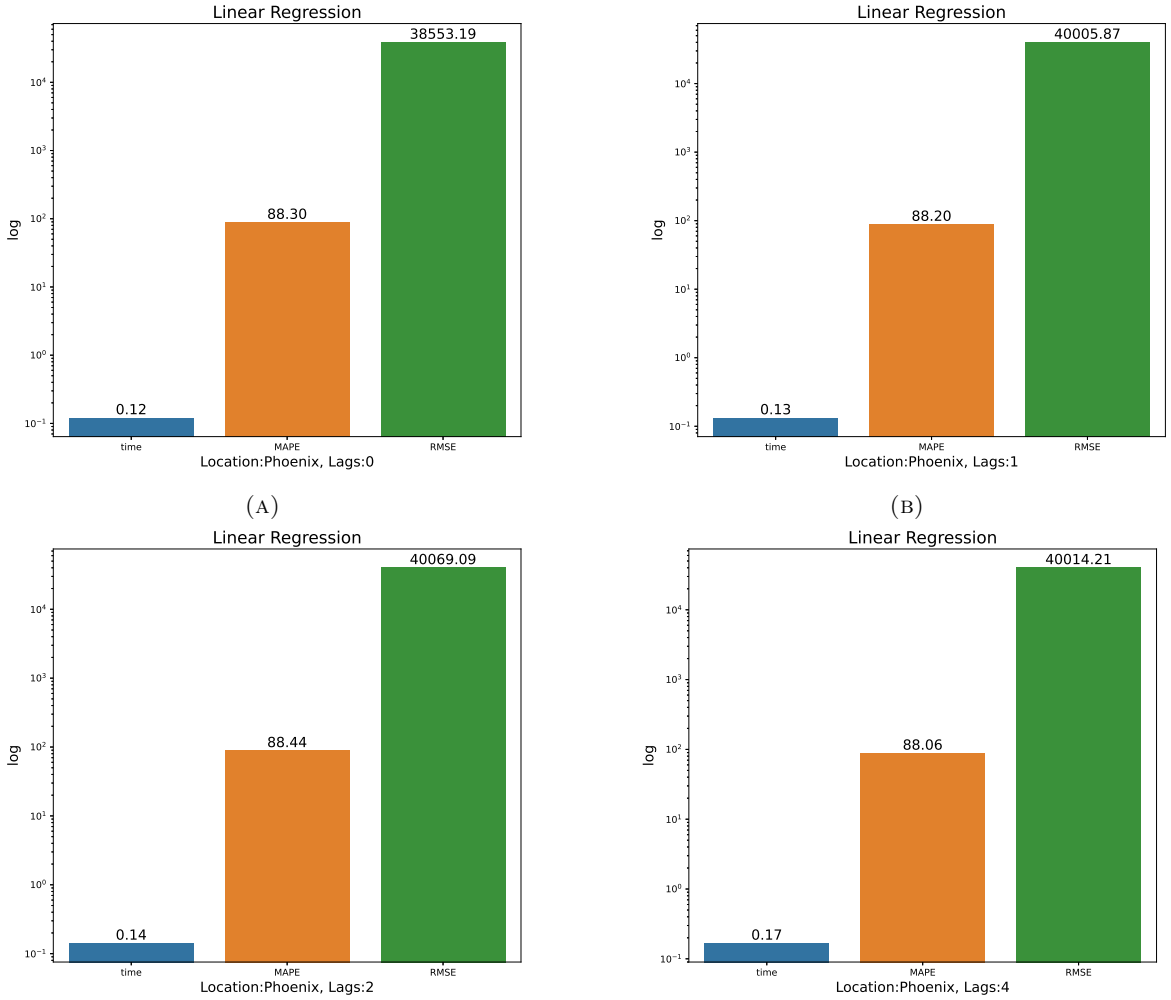
(D)

In Figure 5.1 we see barplots for the time, MAPE and RMSE metrics on the Denver dataset. It has to be mentioned that the use of lag features lowers the error rates, although the number of lags does not seem to make any remarkable difference.



(A) (B) (C) (D)

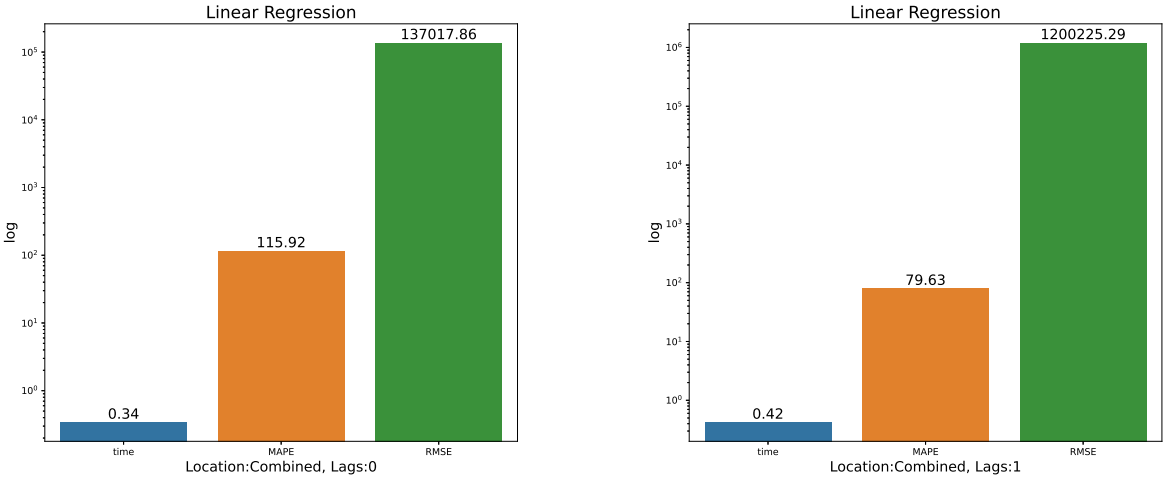
In Figure 5.2 we focus on the Minneapolis dataset. The error rates in general are lower than in the Denver dataset but the result is still not interesting for good predictions. Lag features do not really influence the error in this case.



(A) (B)

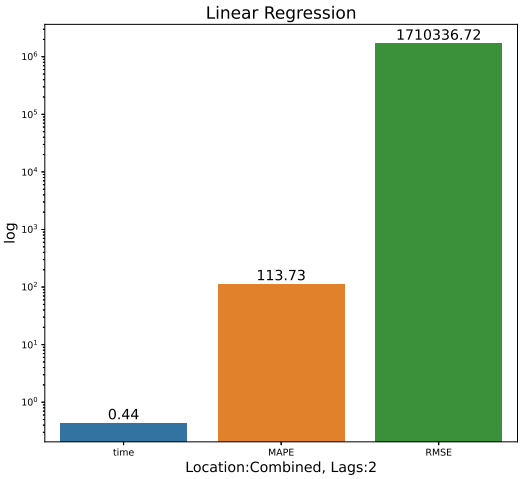
(C) (D)

In the Phoenix case in Figure 5.3 the algorithm's behavior is similar to the Minneapolis case. The error rates are the same and the lag features seem to have no influence in the results.

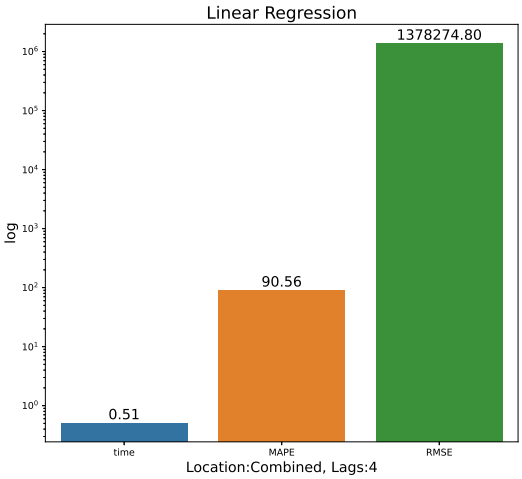


(A)

(B)



(C)



(D)

When testing all the data combined in one dataset, the results are not so positive as seen in Figure 5.4. In particular the error values are between those in Minneapolis/Phoenix cases where the errors are comparatively low and those in Denver where the errors values are high.

The execution time is surprisingly low. On the other hand R^2 is taking big negative values meaning that the algorithm is not appropriate for the problem. This conclusion is also established from the error values that are far bigger than expected. The number

of lag features does not seem to affect the results. The location seems to have stronger influence, for example the Denver dataset is more difficult than the other two locations.

5.3 Kernel Regression

The Kernel Regression implementation is based on the scikit-learn kernelRidge library [17]. On the other hand the the Gaussian Kernel Regression is based on a custom library that builds a kernel based on the normal distribution. Both algorithms need a lot of calculations meaning a lot of disk space. For that purpose parts of the datasets were used in this set of experiments. Specifically only 1000 datapoints of each datasets were used.

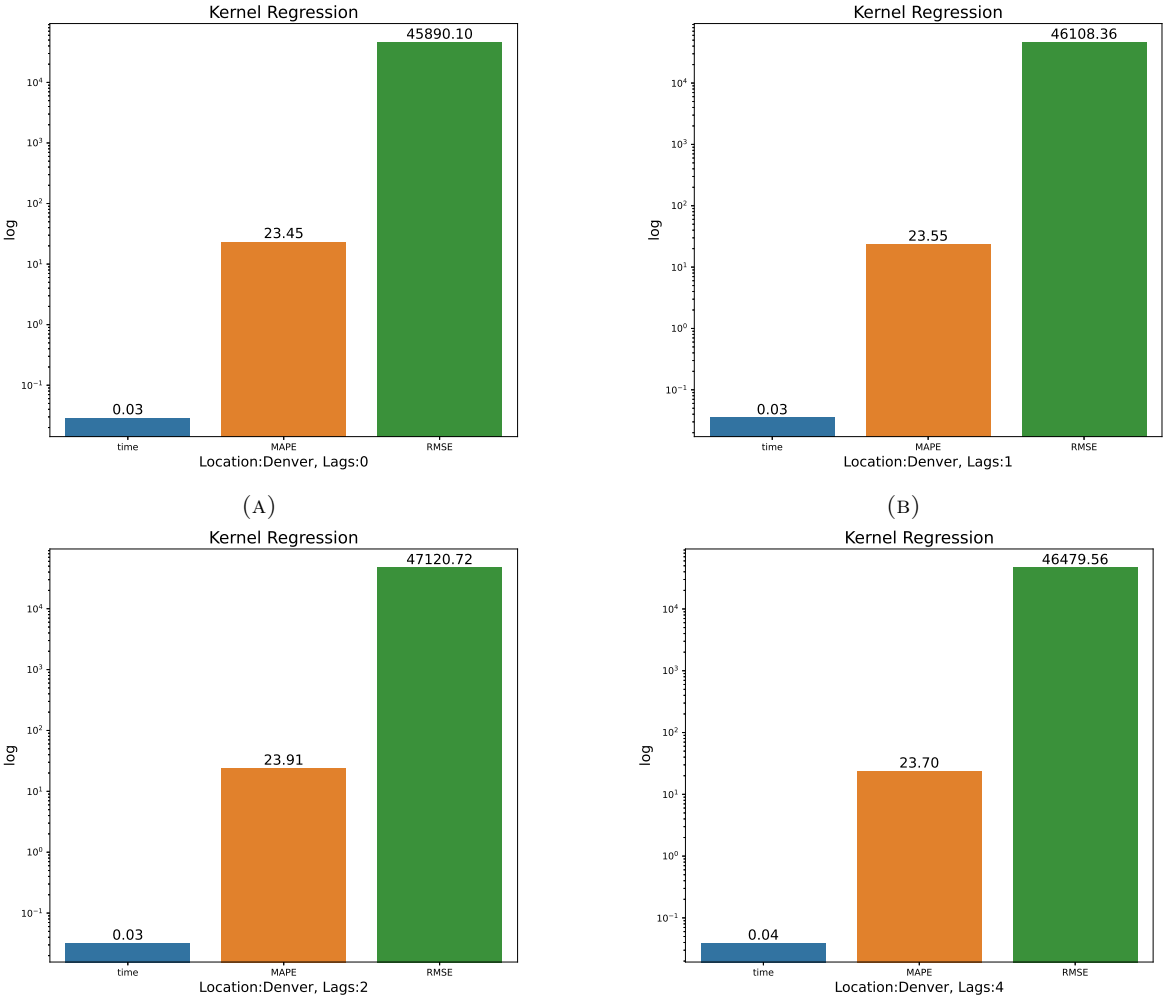
As in Linear Regression in the previous section, the experiments are divided in 4 cases that differ on the dataset used. Each case has 4 subcases for 0,1,2 and 4 lag features respectively. The only difference is that instead of using the whole dataset on each case, only 1000 datapoints were used on each. In the case of the combined dataset, a mixture of Denver, Minneapolis and Phoenix games was picked summarizing in 1000 datapoints. The selection is made by picking random datapoints.

Kernel Regression - Denver Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.02892112732	0.03490591049	0.03191494942	0.03886342049
MAPE	23.45241761	23.55353204	23.91201191	23.70117376
RMSE	45890.10496	46108.36257	47120.71903	46479.5607
R^2	$-9.7 * 10^{10}$	$-11.6 * 10^{10}$	$-11.7 * 10^{10}$	$-1.14 * 10^{11}$

Kernel Regression - Phoenix Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.06083798409	0.05781841278	0.06083774567	0.07280421257
MAPE	96.94789734	96.39241762	97.10063911	97.74946171
RMSE	43463.34958	43112.36615	43416.57335	43711.84864
R^2	$-13.5 * 10^{10}$	$-13.3 * 10^{10}$	$-13.4 * 10^{10}$	$-13.8 * 10^{10}$

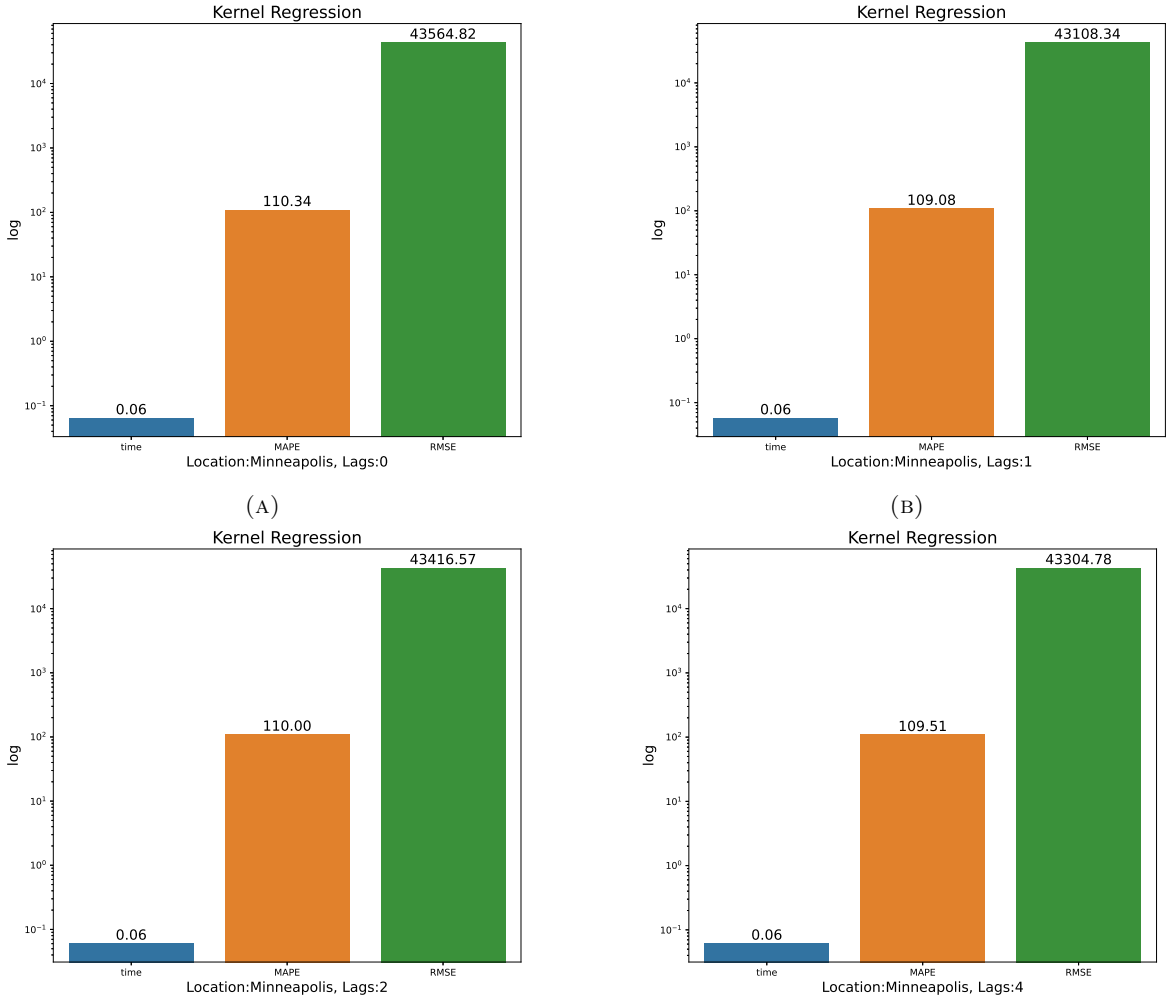
Kernel Regression - Minneapolis Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.06482481956	0.05781412125	0.06083774567	0.06183481216
MAPE	110.336603	109.0767338	110.0034238	109.513759
RMSE	43564.8169	43108.3365	43416.57335	43304.77903
R^2	$-13.9 * 10^{10}$	$-13.7 * 10^{10}$	$-13.4 * 10^{10}$	$-13.2 * 10^{10}$

Kernel Regression - Combined Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.06482553482	0.05983972549	0.06083631516	0.06083726883
MAPE	45.40966703	45.31844791	45.25617342	45.48922863
RMSE	43208.1993	43160.6158	43141.20863	43306.35752
R^2	$-13.6 * 10^{10}$	$-13.4 * 10^{10}$	$-13.2 * 10^{10}$	$-13.5 * 10^{10}$



(A) (B) (C) (D)

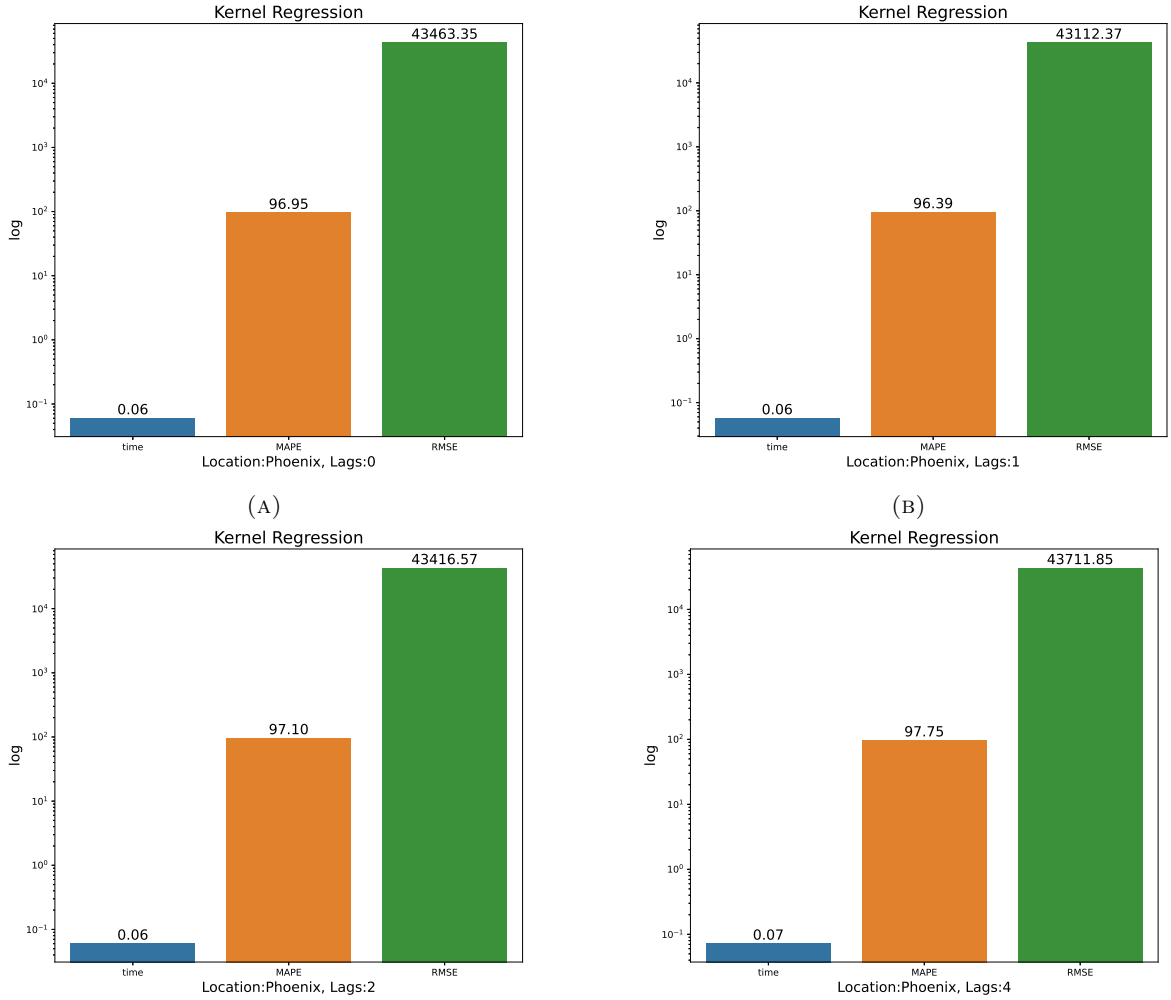
In this set of test cases, the Denver dataset was used. Neither the use of lag features nor their number has influence on the error rates. As seen in Figure 5.5 the results are almost identical in all tests.



(A) (B)

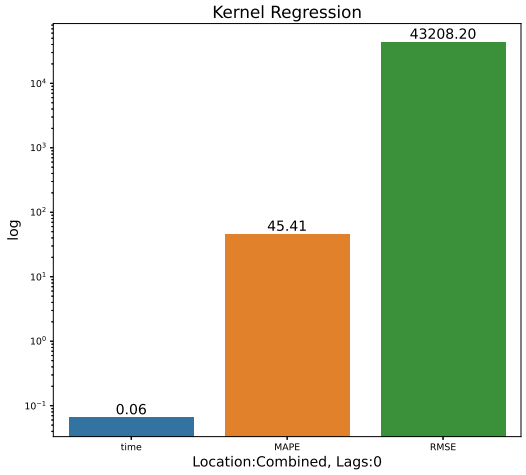
(C) (D)

In Figure 5.6 the Minneapolis dataset is examined. Neither the use of lag features nor their number has influence on the error rates in contrast with Linear Regression where this dataset was ”easier” for the algorithm.

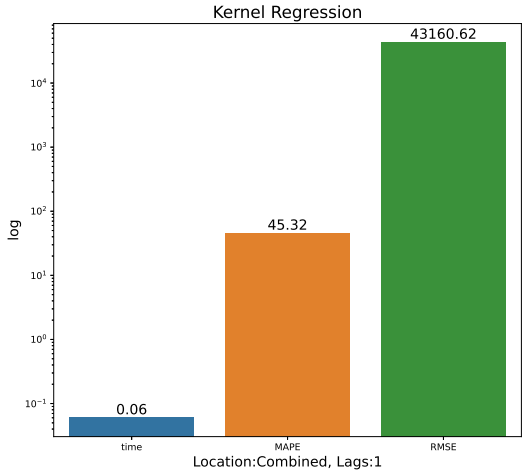


(A) (B) (C) (D)

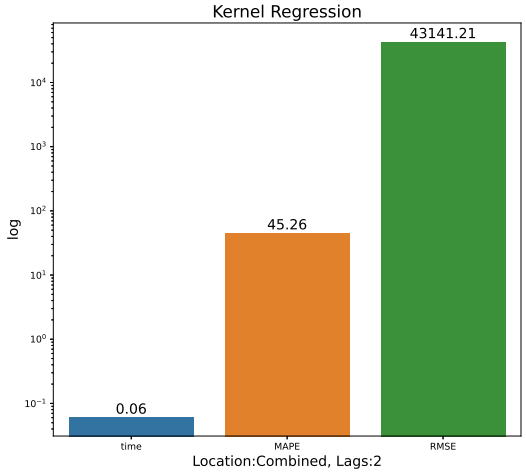
In Figure 5.7 the Phoenix dataset is examined. The lag features make no difference in the final result and the error rates remain in the same level as in the previous cases.



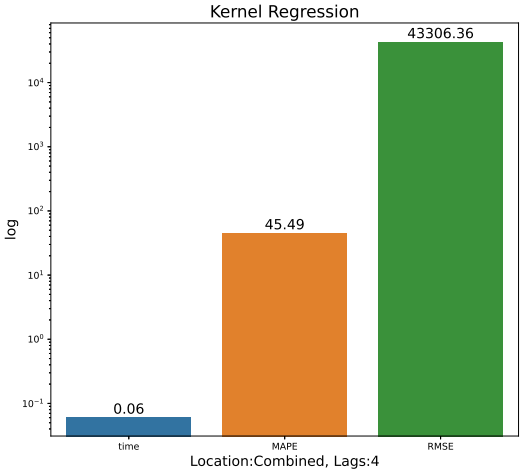
(A)



(B)



(C)



(D)

However the datapoints in the case of Figure 5.8 are taken from games with weather data from different regions the errors remain the same and the lag feature make no difference in the result.

The execution time value is surprisingly low, although it is not taken into consideration because the datapoints were fewer in Kernel Regression than in the other algorithms. On the other hand R^2 is taking big negative values meaning that the algorithm is not appropriate for the problem. This conclusion is also established from the error values

that are far bigger than expected. Neither the number of lag features nor the the location of the dataset seem to affect the result in a betterment.

5.4 k-Nearest Neighbor

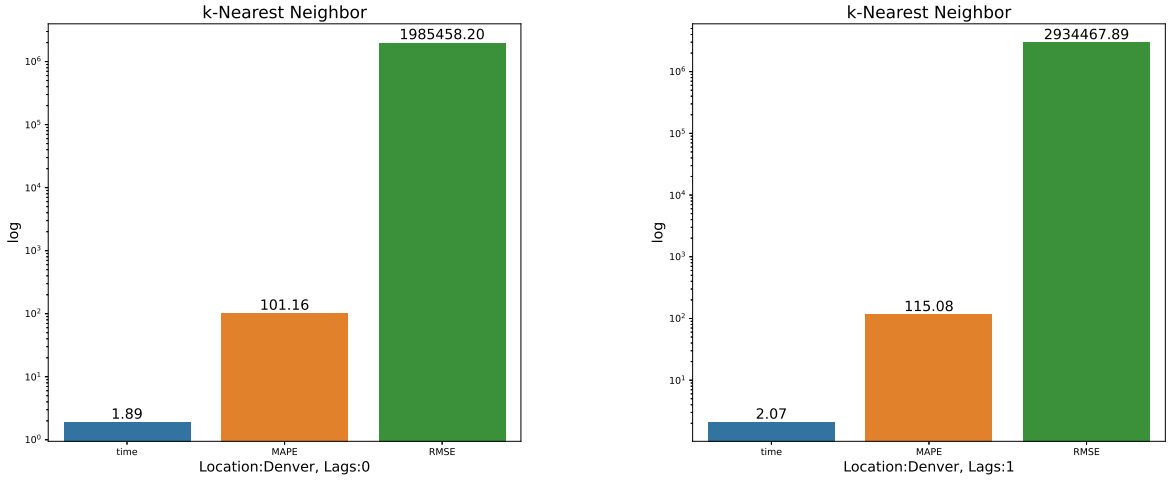
The kNN algorithm implementation is based on the scikit-learn kNeigbhorsRegressor library [17] . The only parameter that was not set by default was the number of the neighbors that was set to 5 after experiments. The bigger the number of neighbors the bigger the number of data that is taken into account in each prediction.

k-Nearest Neighbor (k=5) - Denver Dataset				
Number of Lags	0	1	2	4
Time in seconds	1.890327692	2.073673725	2.143529654	2.389936686
MAPE	101.1554537	115.0820331	100.9810906	91.65961276
RMSE	1985458.199	2934467.888	2684891.427	2494569.154
R^2	$-4.99 * 10^{10}$	$-8.9 * 10^{10}$	$-11.1 * 10^{10}$	$-9.57 * 10^{10}$

k-Nearest Neighbor (k=5) - Phoenix Dataset				
Number of Lags	0	1	2	4
Time in seconds	1.270819664	1.406042576	1.644325972	2.187078714
MAPE	88.20591036	88.24906748	88.21756613	88.07735868
RMSE	40012.43432	40213.46465	40211.13649	40128.10281
R^2	$-10.7 * 10^{10}$	$-11.1 * 10^{10}$	$-11 * 10^{10}$	$-10.98 * 10^{10}$

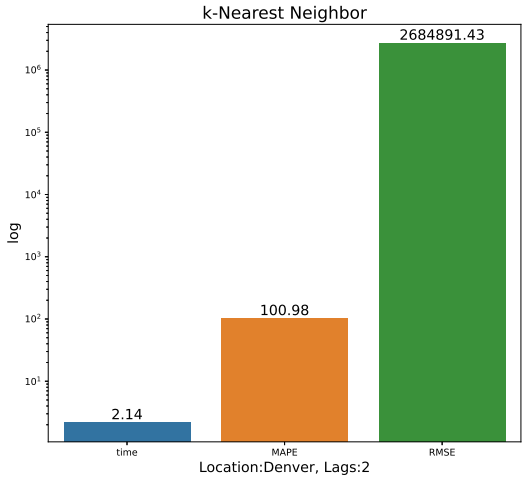
k-Nearest Neighbor (k=5) - Minneapolis Dataset				
Number of Lags	0	1	2	4
Time in seconds	0.06482481956	0.05781412125	0.06083774567	0.06183481216
MAPE	113.6418978	113.7214863	113.7195862	113.5416415
RMSE	43564.8169	43108.3365	43416.57335	43304.77903
R^2	$-17.7 * 10^{10}$	$-17.5 * 10^{10}$	$-17.6 * 10^{10}$	$-17.28 * 10^{10}$

k-Nearest Neighbor (k=5) - Combined Dataset				
Number of Lags	0	1	2	4
Time in seconds	18.67766333	10.30611205	10.64899993	12.37453914
MAPE	90.00540394	80.42891163	104.0111056	86.00772028
RMSE	829837.8594	1209145.64	1639283.392	1308151.061
R^2	$-3.44 * 10^{10}$	$-9.16 * 10^{10}$	$-9.9 * 10^{10}$	$-1.06 * 10^{10}$

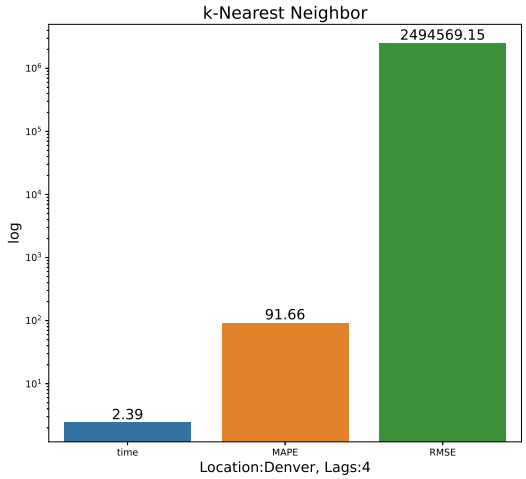


(A)

(B)

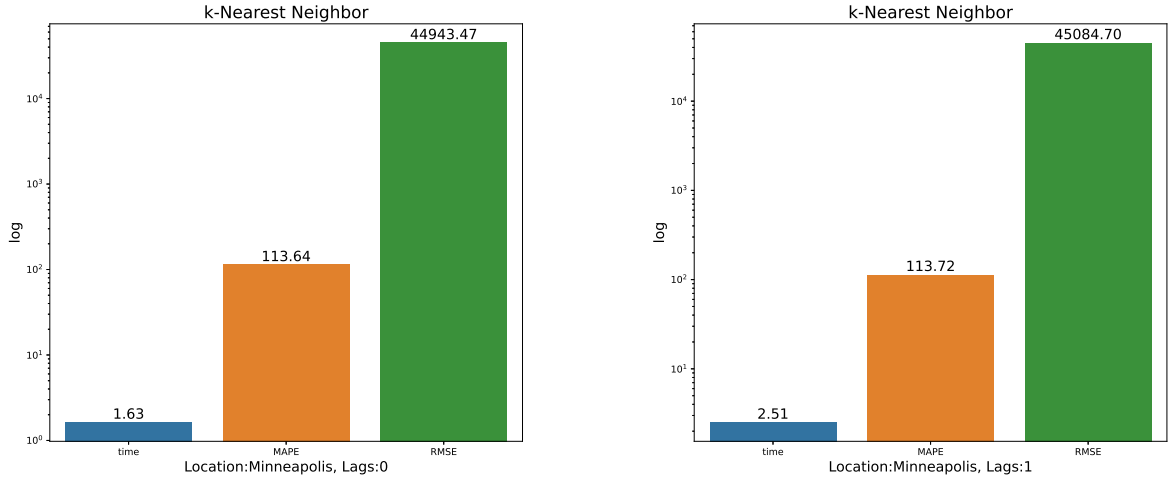


(C)



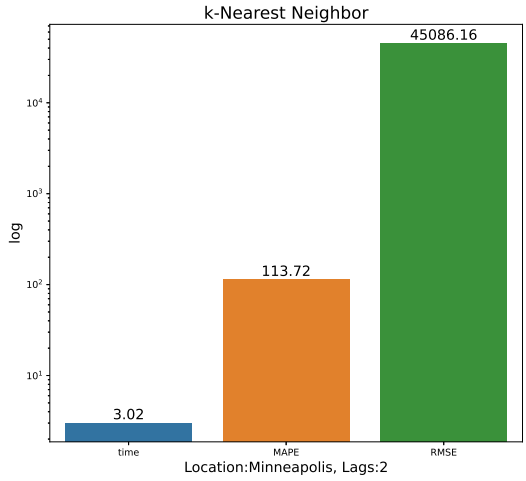
(D)

In Figure 5.9 kNN was implemented with the Denver dataset. The results show that RMSE is not similar to MAPE as in the previous implementations. Based on theory, this could mean that the predictions have bigger deviation in this case.

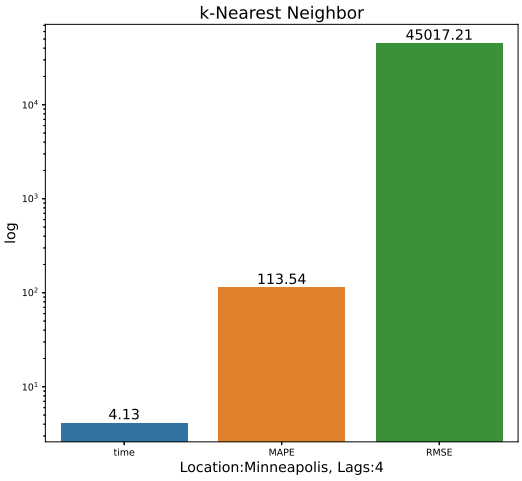


(A)

(B)

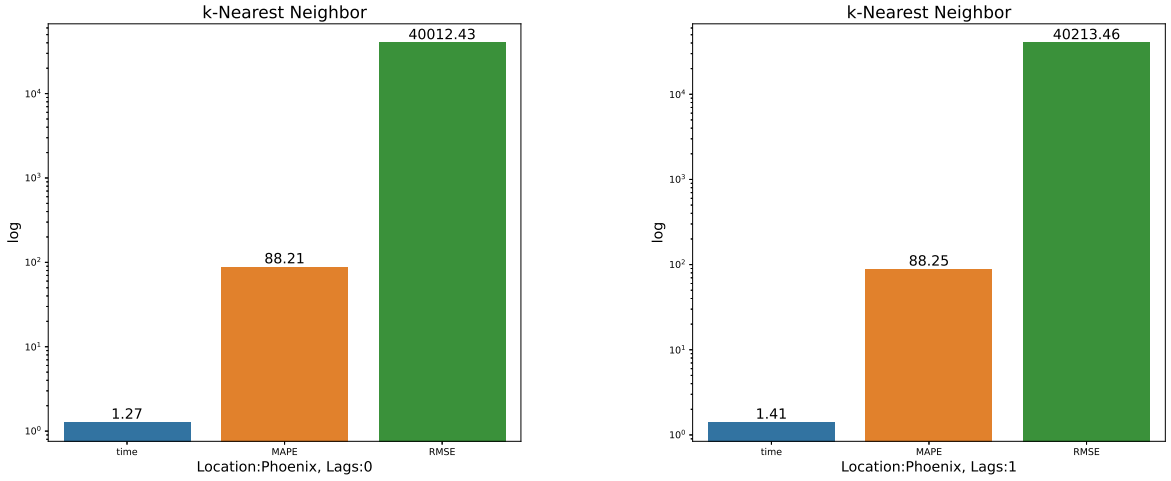


(C)



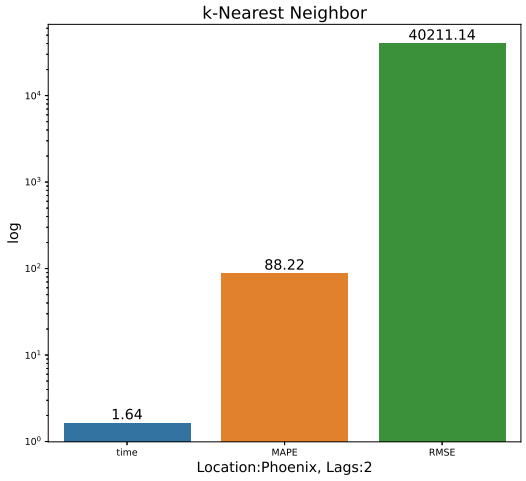
(D)

In the Minneapolis case as shown in Figure 5.10 the error rates are similar with each other. On the other hand the lag features bring no improvement in the error results.

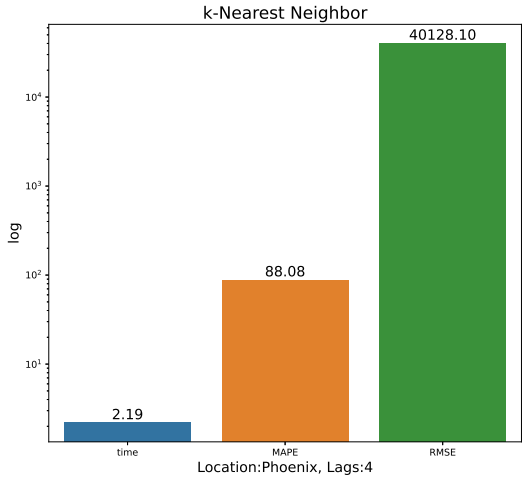


(A)

(B)

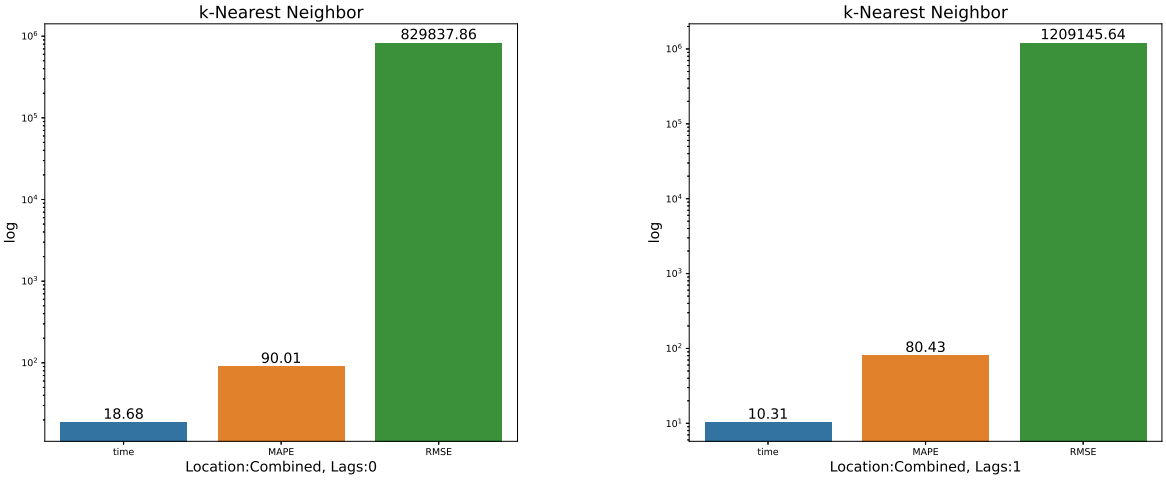


(C)



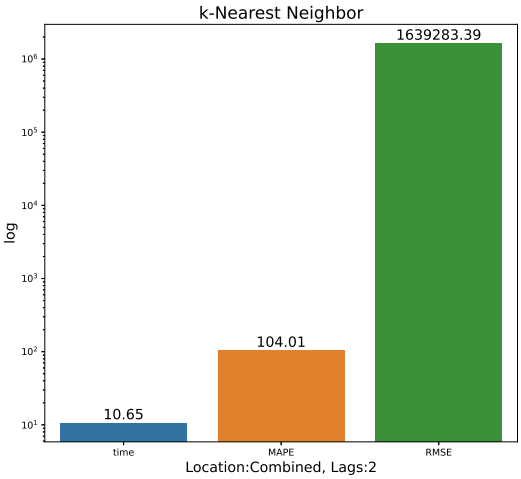
(D)

As shown in Figure 5.11, Phoenix dataset is the "easiest" case for this the kNN algorithm. Additionally, lag features make no impact to the error results.

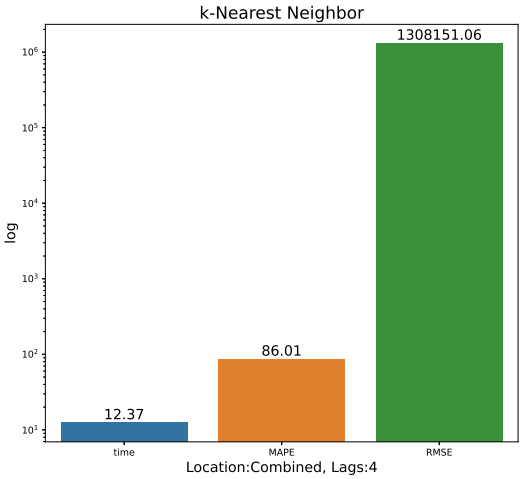


(A)

(B)



(C)



(D)

In Figure 5.12 we can see that RMSE takes really higher values than MAPE when using kNN with the dataset containing datapoints from combined locations. Based on theory, this could mean that the predictions have bigger deviation in this case.

The execution time value is affordably high. On the other hand R^2 is taking big negative values meaning that the algorithm is not appropriate for the problem. This conclusion is also established from the error values that are far bigger than expected.

Neither the number of lag features nor the the location of the dataset seem to affect the result in a betterment.

5.5 Feed Forward Neural Networks

In order to implement the predictor we constructed a NN with the specific parameters and architectural choices shown below:

- 2x Hidden Layers with 24 Neurons each
- Neurons in Input Layer: 9
- Neurons in Output Layer: 1
- Dropout rate: 0.01
- Batch size: 4
- Epochs: 20
- Activation function: ReLU
- Loss function: Mean Square Error (MSE)
- Optimizer: Adam

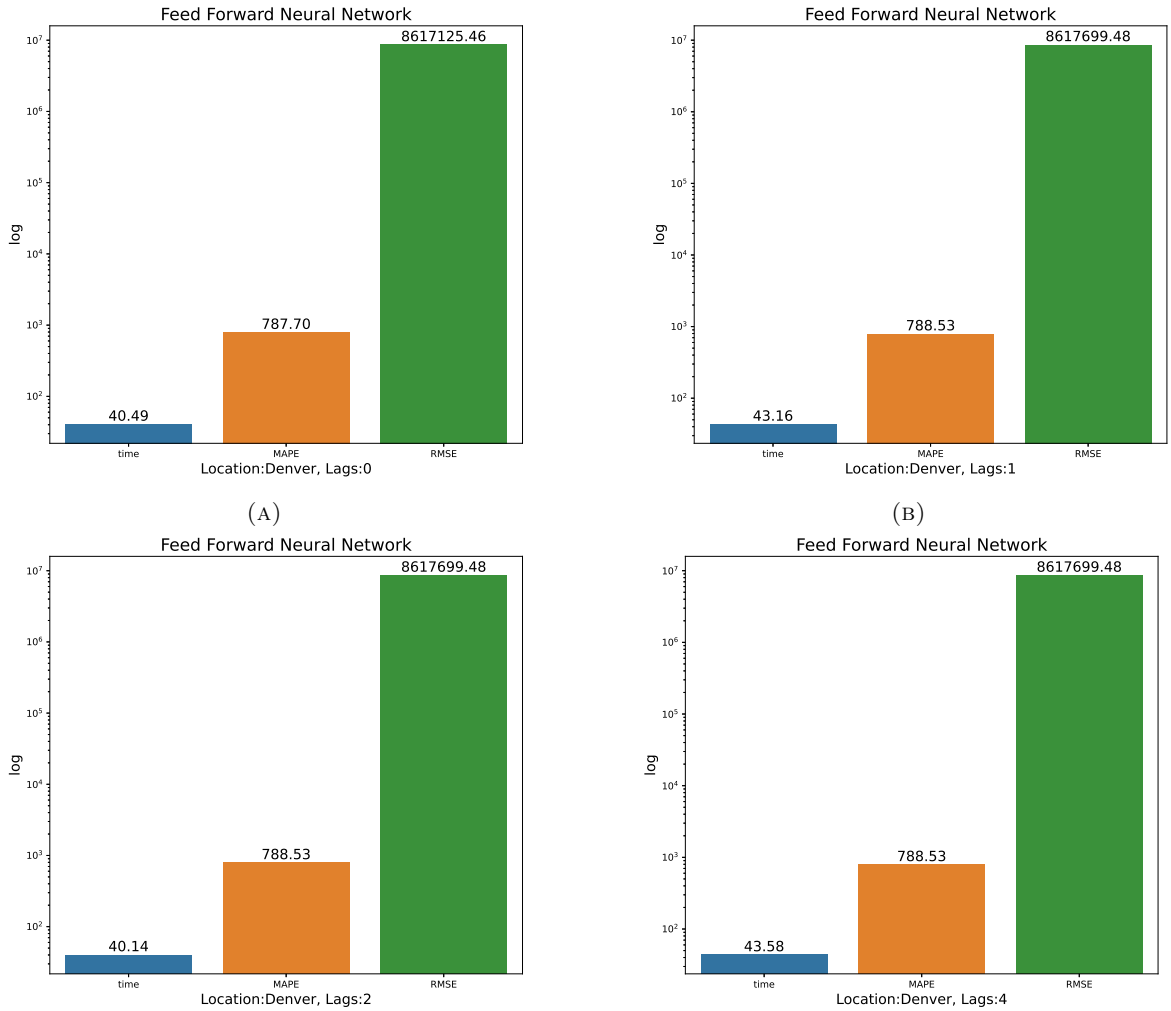
We trained our predictor over this NN with the datasets of the final's games we mentioned before. We used 90% of each dataset for training and the rest 10% for validation. After each dataset training the trained model is saved and before each new training it is loaded from the disk. Apart from run-time in seconds, we also take into account the values of Mean Absolute Percentage Error (MAPE), Rooted Mean Squared Error (RMSE) and R^2 in all of our regression tests.

Feed Forward Neural Networks - Denver Dataset				
Number of Lags	0	1	2	4
Time in seconds	40.49097848	43.15828609	40.13764596	43.57588983
MAPE	787.6988213	788.5284147	788.5284103	788.5284147
RMSE	8617125.456	8617699.477	8617699.476	8617699.477
R^2	-0.03100420	-0.03114166	-0.0311416628	-0.03114166

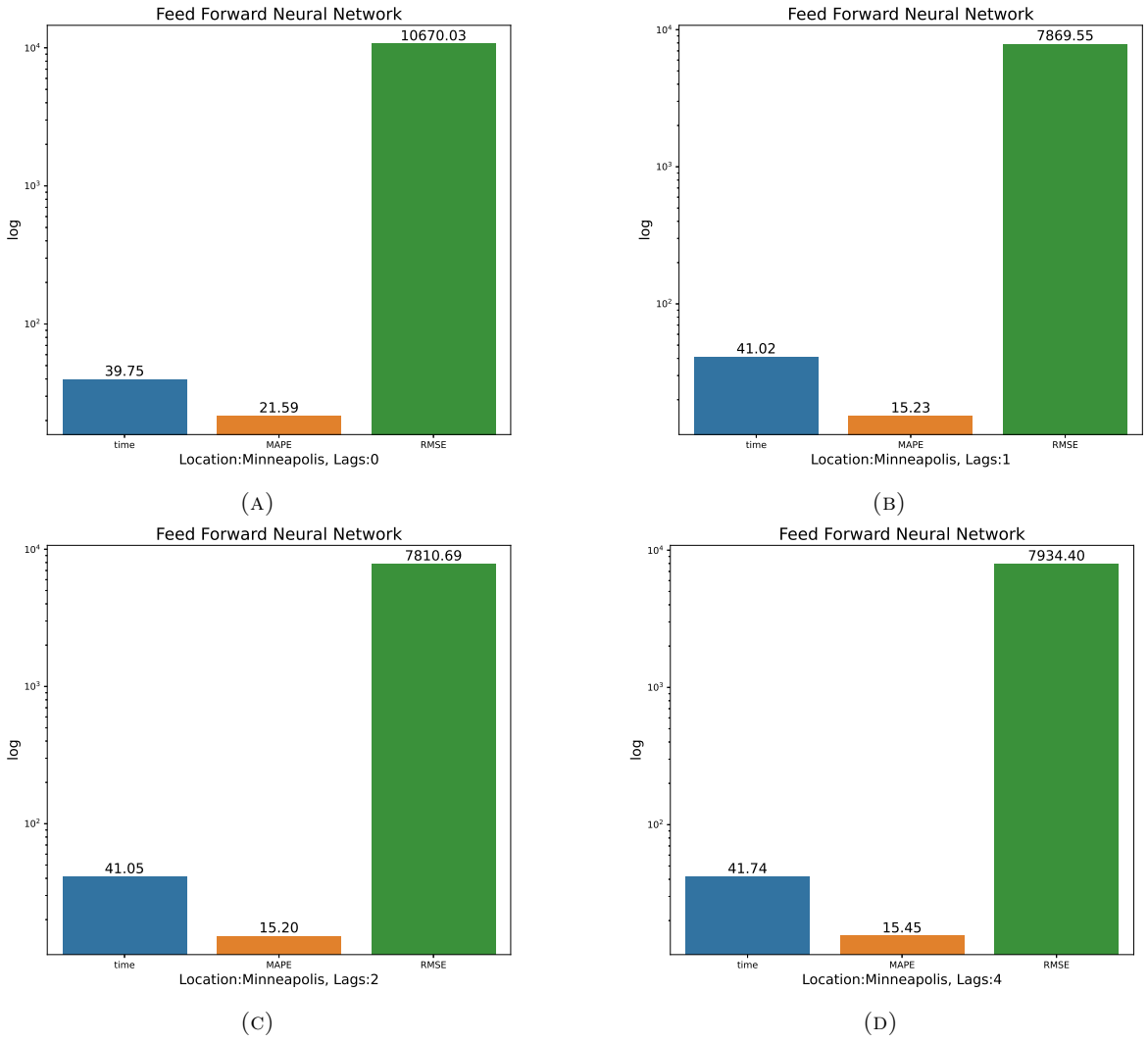
Feed Forward Neural Networks - Phoenix Dataset				
Number of Lags	0	1	2	4
Time in seconds	38.9809854	40.10076928	40.75147986	40.77733421
MAPE	15.66595918	12.00935788	12.01431429	12.21444924
RMSE	8746.375705	6866.879798	6854.433279	6959.471637
R^2	0.5871592541	0.7455249624	0.7464466206	0.7386160991

Feed Forward Neural Networks - Minneapolis Dataset				
Number of Lags	0	1	2	4
Time in seconds	39.75426078	41.02068686	41.04674196	41.74451375
MAPE	21.59211174	15.22535977	15.20030723	15.44645988
RMSE	10670.03115	7869.551916	7810.690478	7934.404433
R^2	0.4020565181	0.6747414129	0.6795888525	0.6693584497

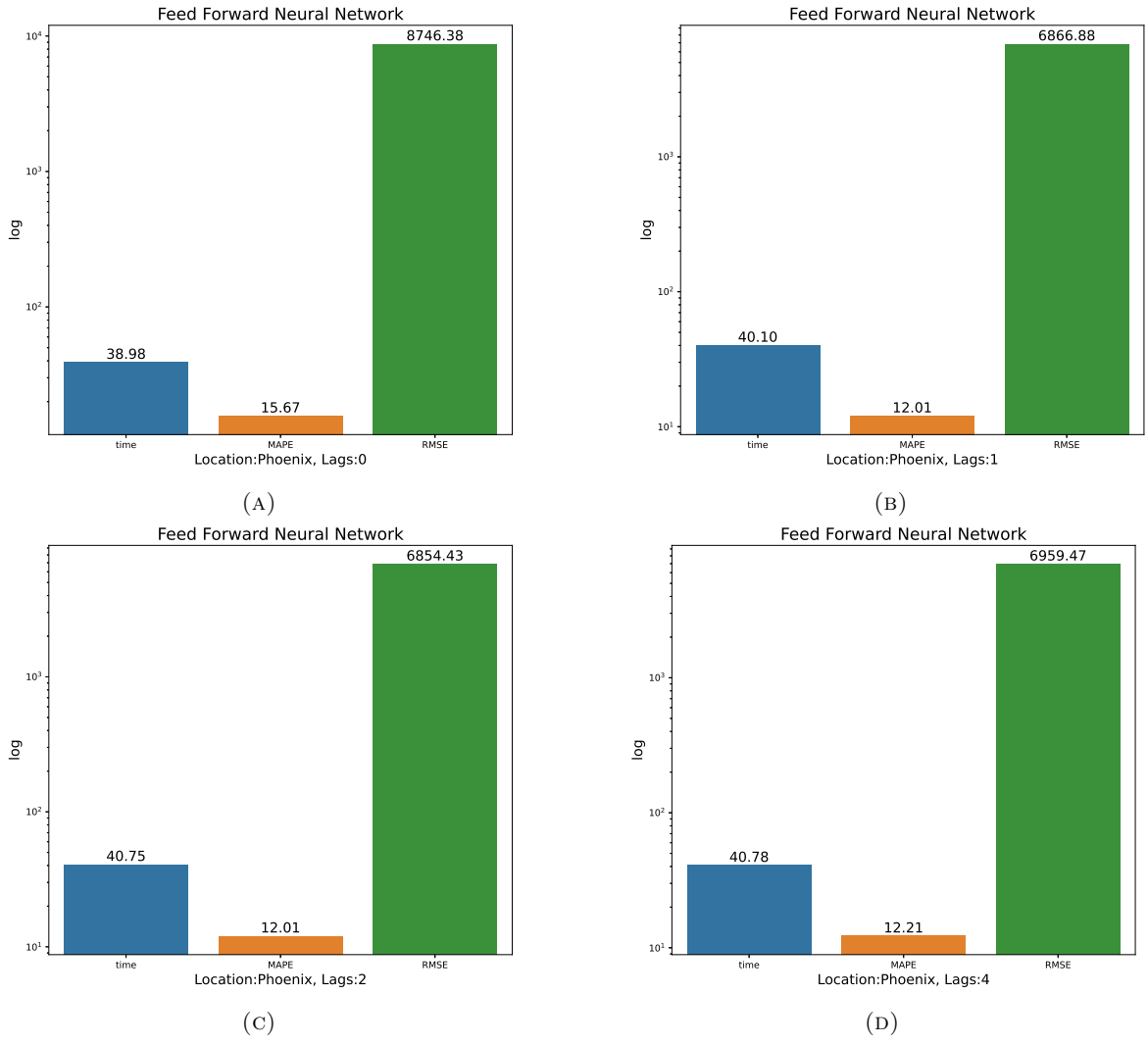
Feed Forward Neural Networks - Combined Dataset				
Number of Lags	0	1	2	4
Time in seconds	132.4048605	122.1000404	130.2168057	134.2977023
MAPE	30.29053432	12.9430265	11.8170175	11.99306198
RMSE	30996.79874	14340.57089	13555.45095	13659.0026
R^2	-4.382113205	-0.152	-0.029	0.0451



In Figure 5.13 the error rates, for Feed Forward Neural Networks on the Denver dataset, are too high. On the other hand the R^2 values shown in the table are close to 0, which is the best value seen until now. Lag features do not seem to influence the result.



In figure 5.14 we can see the best results so far. The error rates are comparatively low and the lag features seem to have influence in this success. The R^2 values are close to 1 which the target value of this metric.



(A) (B) (C) (D)

As shown in Figure 5.15 there is another successful experiment where the error rates are low and the first lag feature lowers them even more. Adding more lag features does not seem to improve matters.

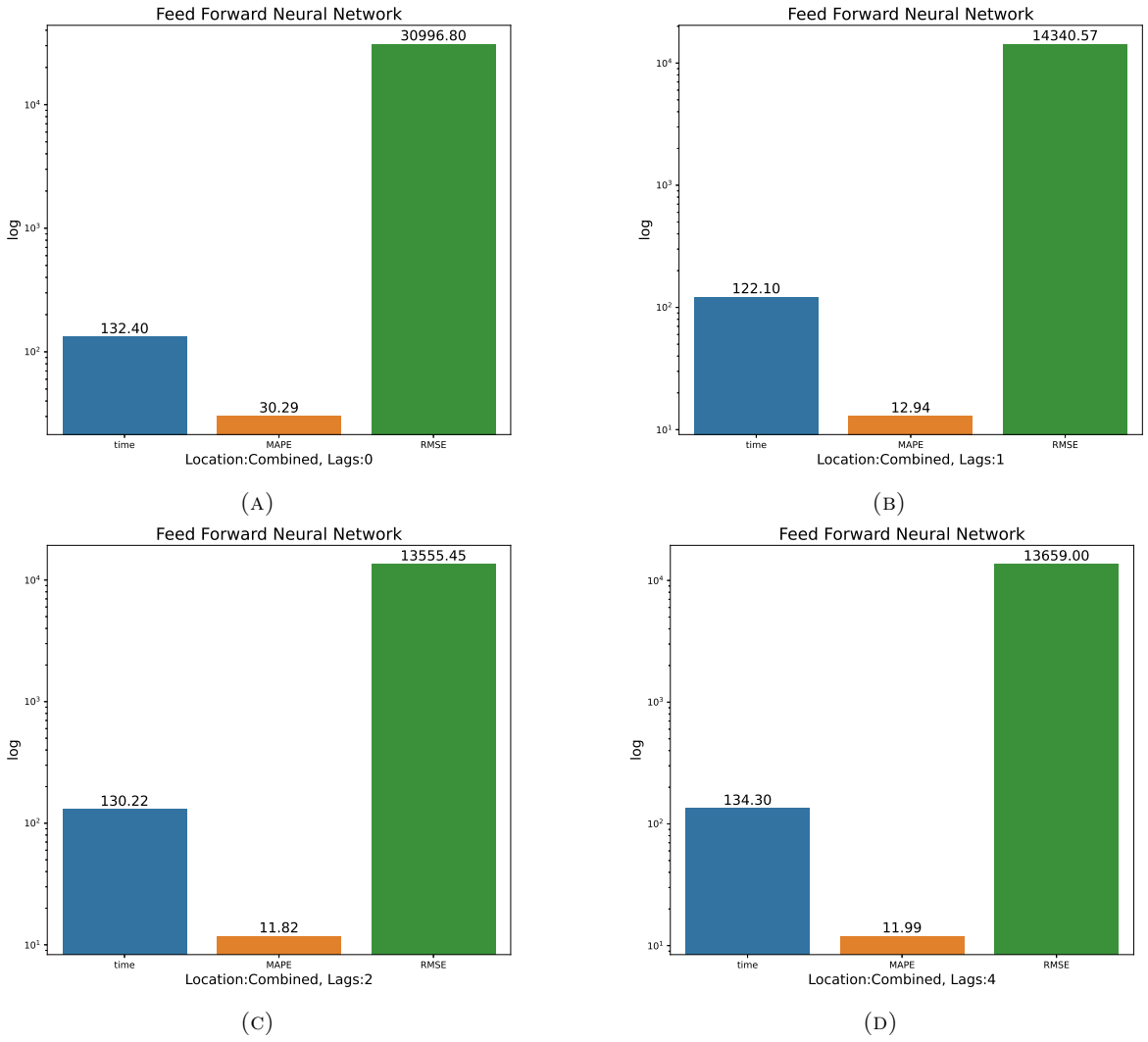


Figure 5.16 depicts the results of the combined dataset through FFNN. It is surprising that the error rates are comparatively lower than in the Denver dataset but still too high compared with the 2 other datasets. The execution time is also really high but that was expected due to bigger number of datapoints.

The value of R^2 is between 0 and 1 meaning that, based in theory, the algorithm works successfully and can relate the features with the target variable. The location seems to affect the results, because in the experiments that use datapoints only from Denver

the errors are much bigger than the experiments with the other locations. Considering the lag features, experiments with 2 and 4 lag features seem to have the smallest errors. However, increasing the lag features to more than 4 (as we tried in additional experiments), increases the execution time but does not decrease the errors. The execution time takes prohibitively high values. This means that we should either pre-train our model or train it online with a smaller dataset. Another way to combat some complexity would be to make our Neural Network less complex, but that would most probably cost us in accuracy.

5.6 Recurrent Neural Networks

In order to implement the predictor we constructed a Recurrent NN using the LSTM algorithm with specific architecture:

- 3x Hidden Layers with 24 Neurons each
- Neurons in Input Layer: 9
- Neurons in Output Layer: 1
- Dropout rate: 0.01
- Batch size: 8
- Epochs: 10
- Activation function: ReLU
- Loss function: Mean Square Error (MSE)
- Optimizer: Adam

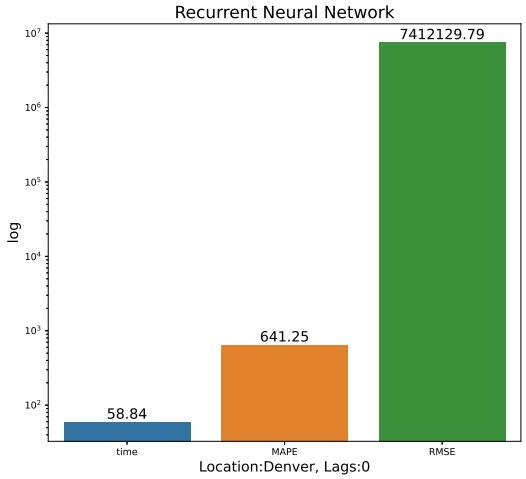
We trained our predictor over this NN with the datasets of the final's games we mentioned before. We used 90% of each dataset for training and the rest 10% for validation. After each dataset training the trained model is saved and before each new training it is loaded from the disk. Apart from run-time in seconds, we also take into account the values of Mean Absolute Percentage Error (MAPE), Rooted Mean Squared Error (RMSE) and R^2 in all of our regression tests.

Recurrent Neural Networks - Denver Dataset				
Number of Lags	0	1	2	4
Time in seconds	58.8414479	59.48851445	59.83600238	59.81850654
MAPE	641.2501379	630.6928129	625.2851191	621.0355086
RMSE	7412129.789	7387129.123	7344749.79	7312129.334
R^2	-0.031004299	-0.031141662	-0.031141662	-0.031141662

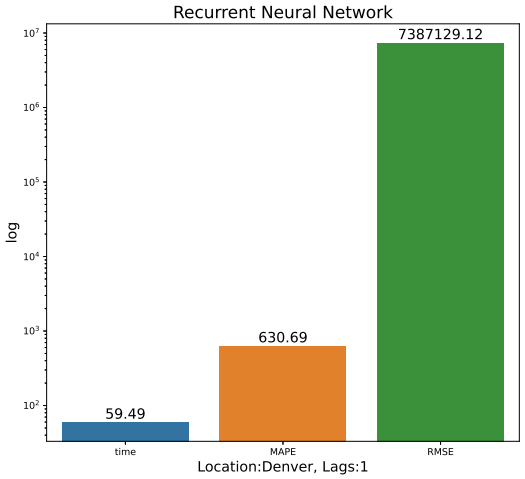
Recurrent Neural Networks - Phoenix Dataset				
Number of Lags	0	1	2	4
Time in seconds	56.53477681	57.07219443	57.64762034	58.17401793
MAPE	12.93810114	9.728833672	9.723125586	9.729327761
RMSE	7956.789448	6566.123445	6498.789887	6547.333559
R^2	-0.6853433563	-0.7500329415	-0.7190772563	-0.7392615246

Recurrent Neural Networks - Minneapolis Dataset				
Number of Lags	0	1	2	4
Time in seconds	56.89601111	57.16239751	57.97746658	58.57463387
MAPE	18.16392356	14.74989204	14.65144122	14.7480879
RMSE	8996.375705	6746.879798	6658.433279	6884.471637
R^2	-0.0319787871	-0.0567520541	-0.0492601775	-0.0307545423

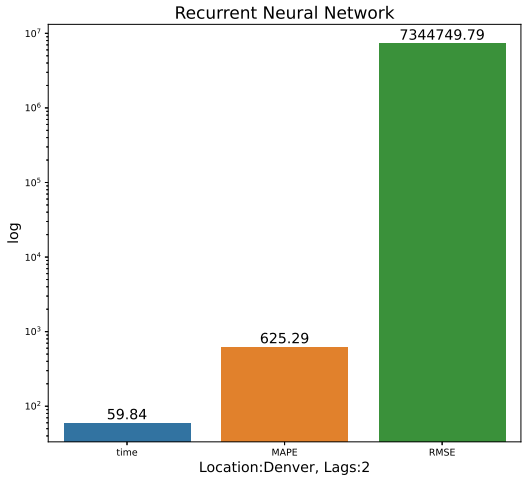
Recurrent Neural Networks - Combined Dataset				
Number of Lags	0	1	2	4
Time in seconds	176.5713931	180.015196	183.5863291	186.9637094
MAPE	24.37575432	11.05935515	10.79769823	10.98042558
RMSE	24117.87496	12254.46617	11157.62516	11448.25842
R^2	-4.381976114	-4.382	-0.0459	-0.0014



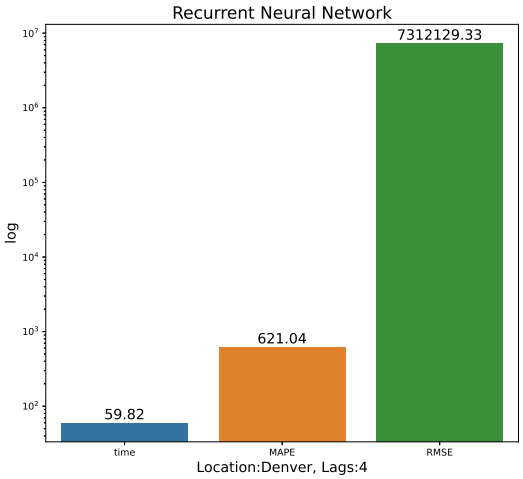
(A)



(B)

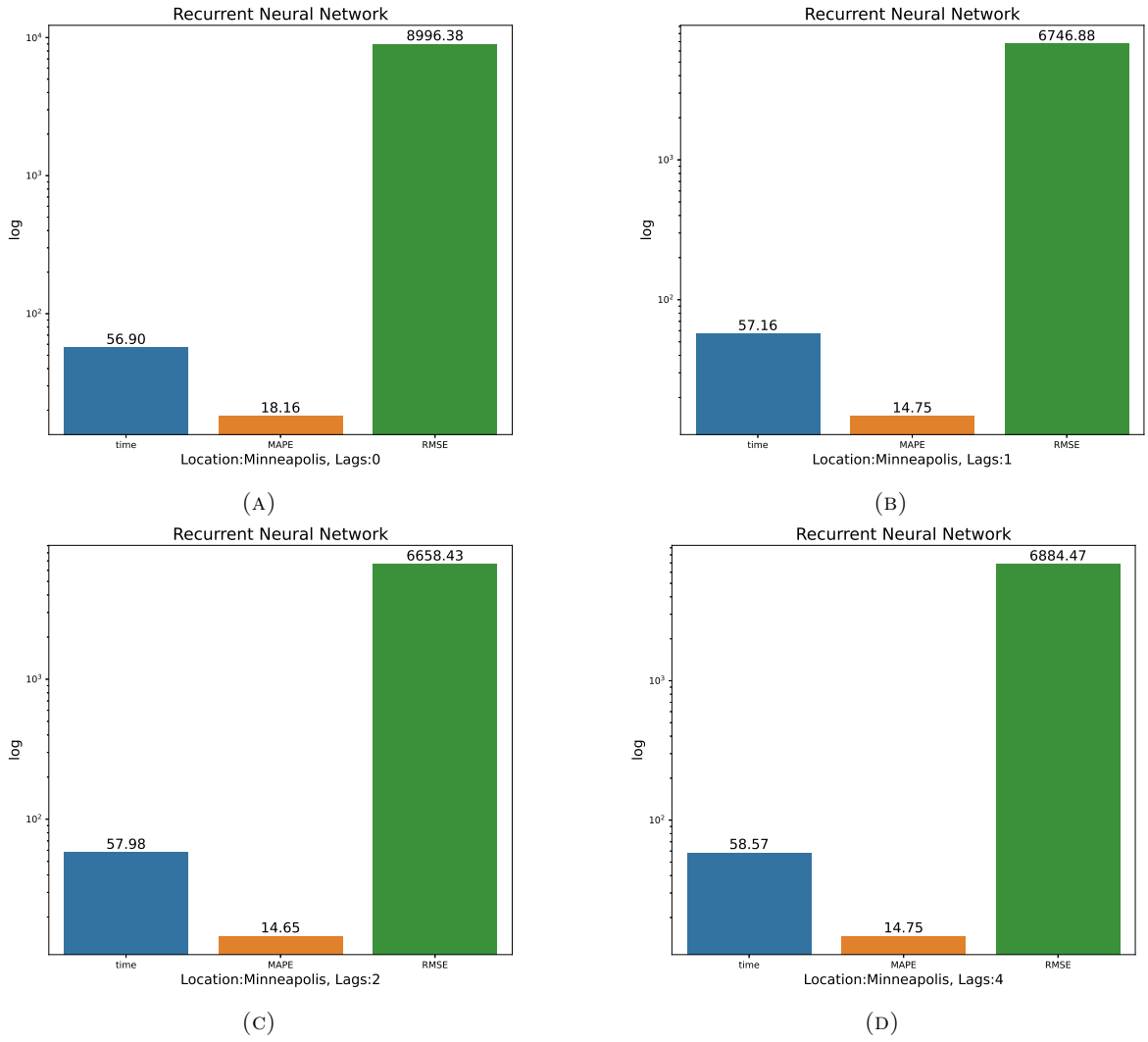


(C)



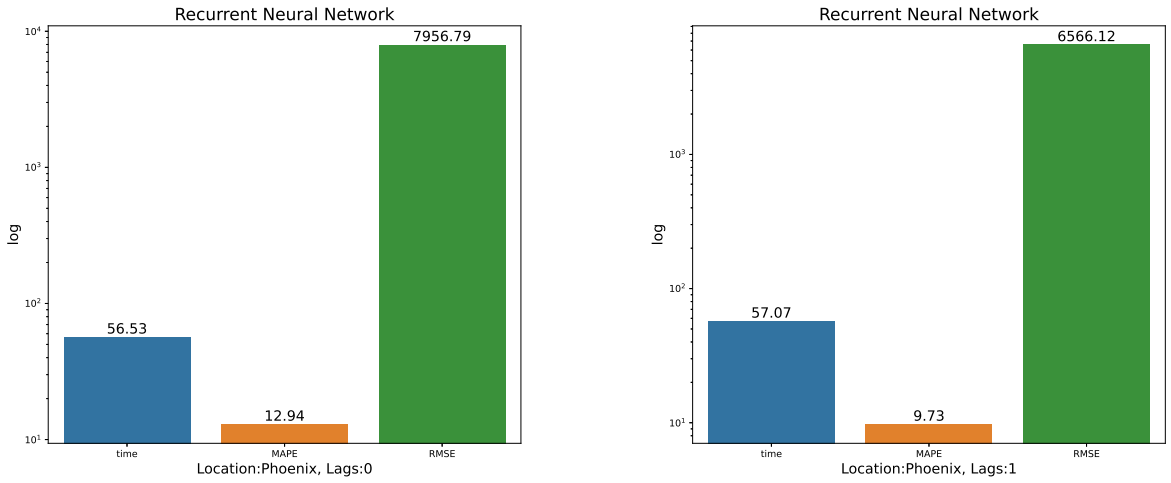
(D)

As seen in figure 5.17 the error rates are really high. RMSE is higher than MAPE meaning that the predictions have big divergence from the real values. Moreover the lag features have no impact in lowering the errors. The R^2 value is slightly below 0.



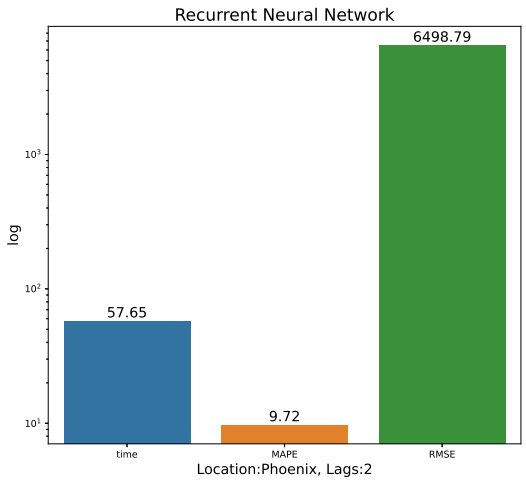
(A) (B) (C) (D)

In Figure 5.18 we can see that Minneapolis dataset is "easier" than Denver for the Recurrent Neural Networks. The errors have comparatively low values and the lag features make an obvious difference to the results. The R^2 value is slightly below 0.

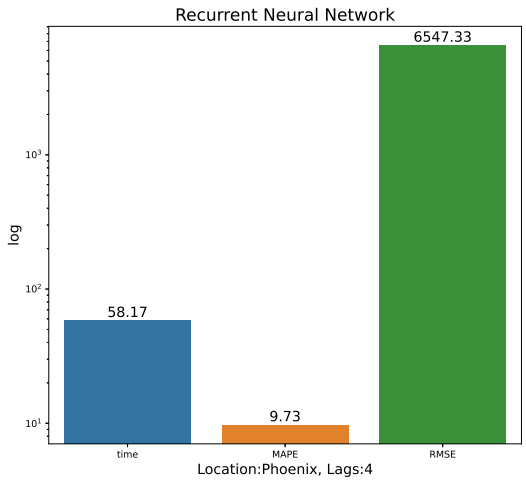


(A)

(B)

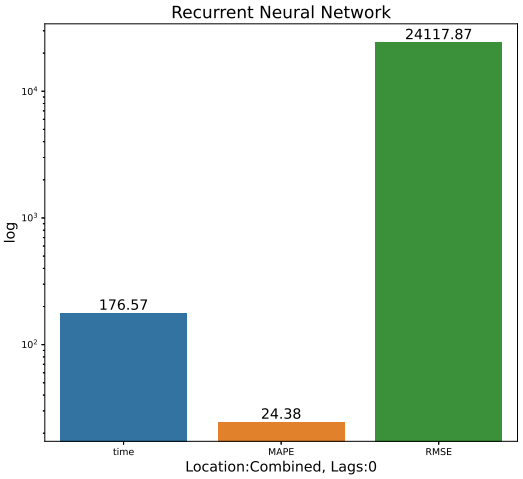


(C)

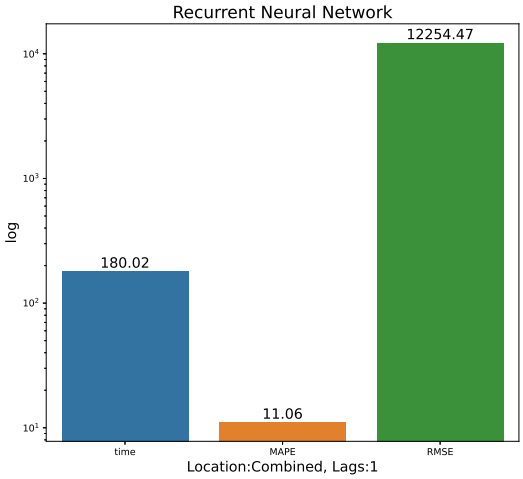


(D)

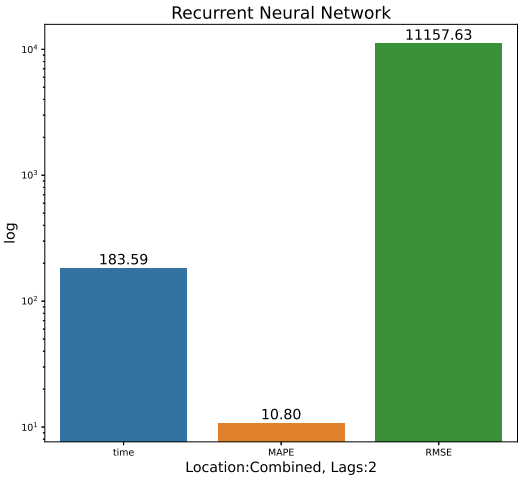
As shown in figure 5.19 Phoenix dataset works very well with the Recurrent Neural Networks. The error values are comparatively low and the lag features lowers them even more. R^2 value is close to -1, which is not great but not very disappointing either.



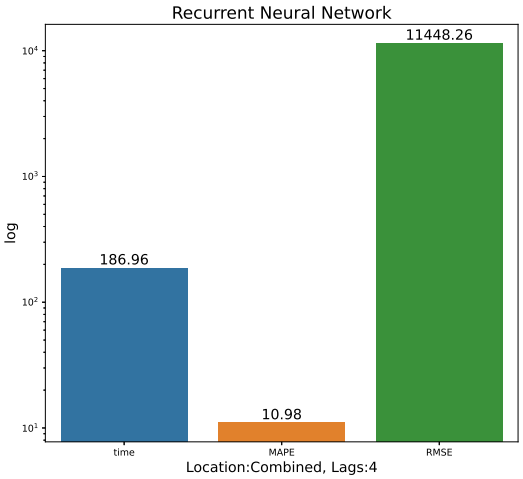
(A)



(B)



(C)



(D)

The value of R^2 is close to 0 meaning that the algorithm works successfully and can relate the features with the target variable. The location seems to affect the results, because in the experiments that use datapoints only from Denver the errors are much bigger than the experiments with the other locations. Considering the lag features, experiments with 2 and 4 lag features seem to have the smallest errors. However, increasing the lag features to more than 4, increases the execution time but does not decrease the errors. The execution time takes prohibitively high values, almost double than that of the FFNN. This means that we should either pre-train our model or train

it online with a smaller dataset. Another way to earn combat time complexity would be to make our Neural Network less complex but that would most probably cost us in accuracy.

5.7 Comparison between the Algorithms

None of the algorithms was as successful as we expected, however some of them have interesting results that could improved with further research. This conclusion emanates from the values of the R^2 metric. This metric had irrelevant values in almost all algorithms except for the FeedForward Neural Networks where it was between 40 and 75 percent, for the non-Denver games.

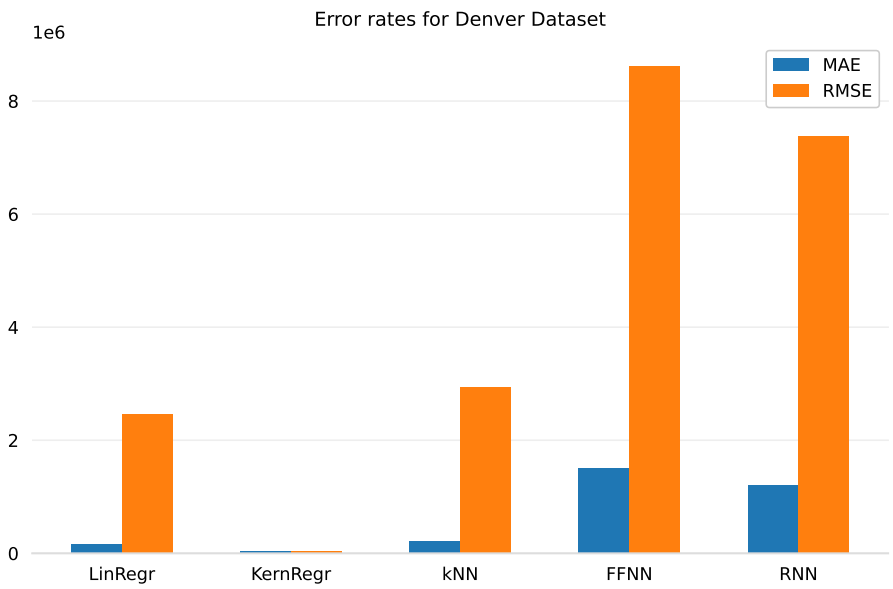


Figure 5.21. Error rates of all the algorithms with the Denver Dataset

Despite that Figure 5.21 fools the eyes, none of the algorithm could approach good predictions for the Denver datasets. Kernel Regression had the best results but these predictions diverge about 90% on average from the real values.

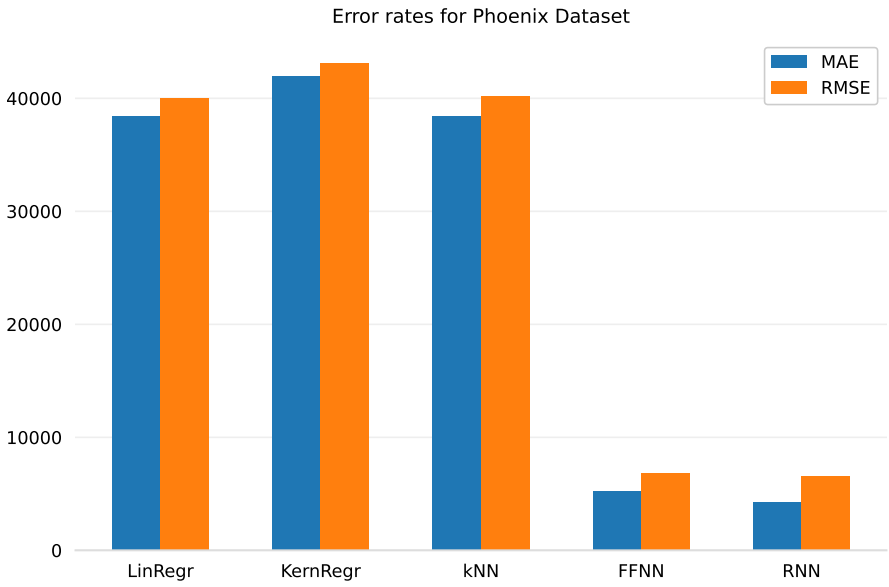


Figure 5.22. Error rates of all the algorithms with the Phoenix Dataset

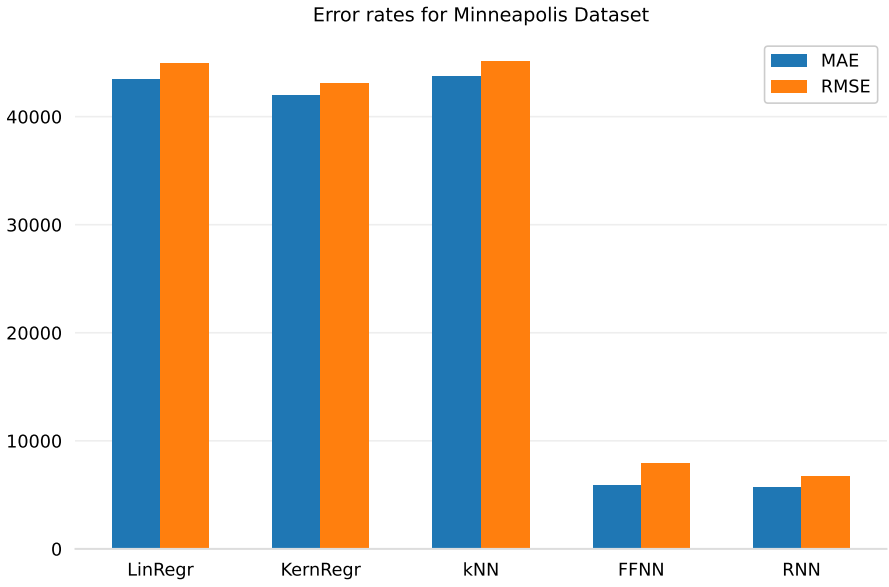


Figure 5.23. Error rates of all the algorithms with the Minneapolis Dataset

In both figures 5.22 and 5.23 the results are clear. Both Deep Learning Algorithms work far better than the Machine Learning Algorithms. The error rates are about 5 times smaller, while in a closer view FFNNs are better in comparison to RNNs.

Figure 5.24. Error rates of all the algorithms with the Combined Dataset

Finally, in Figure 5.24 Deep Learning Algorithms remain reliable, while Machine Learning Algorithms grow even higher error rates.

Chapter 6

Conclusions

This project's target was to create an energy usage predictor module for the TUC-TAC agent for the PowerTAC 2020 tournament. The problem was approached with Machine Learning Regression Algorithms. Neural Network models were also implemented. The input data was relevant to time, date, weather conditions and past usage. Creating an energy predictor with the use of Machine Learning for the Smart Grid was a complex task.

Although the target of this project was not fully accomplished, useful conclusions and intuitions were extracted from the process. Starting from the feature selection and the data analysis our scope has to widen our view instead of focusing only on the weather condition data. Furthermore, a lot of different algorithms were implemented and compared with each other. The results shows that Neural Networks are more suitable than the other regression methods. This was perhaps to be expected for a complex problem such as ours. Possible improvements and additions are going to be discussed in the next paragraph.

6.1 Future Work

Despite the fact that numerous methods were implemented for this project, there is plenty of room for improvements and additions in the project. Starting from the feature selection, selecting features like customer type, customer usage, customer workdays could improve the model of the predictor. This would work like creating an entity for the customers, with the prospect of making the data clearer for the Neural Networks to operate on. Furthermore, the predictor could improve with the addition of extra modules. Net usage is not the only variable that is interesting and useful to predict. More variables, like the buying and the selling price are important for the agent, and thus could be of importance to prediction algorithms. Last but not least hyper-parameter tuning would be crucial. This change is important for reliability reason and could also upgrade the performance.

References

[1] N.S. Altman. ≪An Introduction to Kernel and Nearest Neighbor Nonparamentric Regression≫. en. In: (1991), p. 32.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Springer, 1992.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Ryan R. Curtin et al. ≪mlpack 3: a fast, flexible machine learning library≫. In: *Journal of Open Source Software* 3 (26 2018), p. 726. DOI: 10.21105/joss.00726. URL: https://doi.org/10.21105/joss.00726.

[5] Laurene Fausett. *Fundamentals of Neural Networks*. Pearson, 1994.

[6] Susobhan Ghosh et al. ≪Bidding in Smart Grid PDAs: Theory, Analysis and Strategy≫. In: ().

[7] Susobhan Ghosh et al. ≪VidyutVanika: A Reinforcement Learning Based Broker Agent for a Power Trading Competition≫. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 914–921. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.3301914. URL: https://aaai.org/ojs/index.php/AAAI/article/view/3880 (visited on 07/01/2021).

[8] Susobhan Ghosh et al. ≪VidyutVanika: An Autonmous Broker Agent for Smart Grid Environment≫. en. In: (), p. 6.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[10] Demijan Grgic et al. ≪CrocodileAgent 2018: Robust agent-based mechanisms for power trading in competitive environments≫. en. In: *Computer Science and Information Systems* 16.1 (2019), pp. 105–129. ISSN: 1820-0214, 2406-1018. DOI: 10.2298/CSIS181010040G. URL: http://www.doiserbia.nb.rs/Article.aspx?ID=1820-021418000040G (visited on 07/01/2021).

[11] Geoffrey E. Hinton. ≪Learning in Parallel Networks≫. en. In: (), p. 6.

[12] J. D. Hunter. ≪Matplotlib: A 2D graphics environment≫. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[13] Michael Kamp et al. ≪Efficient Decentralized Deep Learning by Dynamic Model Averaging≫. en. In: *arXiv:1807.03210 [cs, stat]* (Nov. 2018). arXiv: 1807.03210. URL: http://arxiv.org/abs/1807.03210 (visited on 12/09/2019).

[14] Wolfgang Ketter, John Collins, and Mathijs de Weerdt. ≪The 2020 Power Trading Agent Competition≫. en. In: (), p. 47.

[15] Stavros Orfanoudakis. ≪Developing an Autonomous Agent for Automated Electricity Trading≫. en. In: (2020). Senior Undergraduate Diploma Thesis(Master of Engineering), Technical University of Crete.

[16] Serkan Özdemir and Rainer Unland. ≪AgentUDE17: Imbalance Management of a Retailer Agent to Exploit Balancing Market Incentives in a Smart Grid Ecosystem≫. en. In: (), p. 13.

[17] F. Pedregosa et al. ≪Scikit-learn: Machine Learning in Python≫. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[18] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2006.

[19] Sarvapali D Ramchurn et al. ≪Putting the "Smarts" into the Smart Grid: A Grand Challenge for Artificial Intelligence≫. en. In: (), p. 9.

[20] Ansel Y. Rodríguez González et al. ≪A competitive and profitable multi-agent autonomous broker for energy markets≫. en. In: *Sustainable Cities and Society* 49 (Aug. 2019), p. 101590. ISSN: 22106707. DOI: 10.1016/j.scs.2019.101590. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210670718316664 (visited on 07/01/2021).

[21] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. URL: https://doi.org/10.5281/zenodo.3509134.

[22] Jerome Friedman Trevor Hastie Robert Tibshirani. *The Elements of Statistical Learning*. Oxford University Press, 2009.

[23] Daniel Urieli and Peter Stone. ≪An MDP-Based Winning Approach to Autonomous Power Trading: Formalization and Empirical Analysis≫. en. In: ().

[24] Daniel Urieli and Peter Stone. ≪Autonomous Electricity Trading Using Time-of-Use Tariffs in a Competitive Market≫. en. In: ().

Appendix A

Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
DL	Deep Learning
RL	Reinforcement Learning
ANN	Artificial Neural Network
NN	Neural network
FFNN	Feed-Forward Neural Network
MSE	Mean Squared Error
DNN	Deep Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
NLP	Natural Language Processing