2022-08-01

Forecasting Solar Flares Using Shallow And Deep Learning Rechniques

Sumi Dey
University of Texas at El Paso

FORECASTING SOLAR FLARES

USING SHALLOW AND DEEP LEARNING TECHNIQUES

SUMI DEY

Doctoral Program in Computational Science

APPROVED:

Olac Fuentes, Chair, Ph.D.

Amy Wagler, Ph.D.

Laura Boucheron, Ph.D.

Stephen L. Crites, Ph.D.
Dean of the Graduate School

Dedication

to my

Family

with love

FORECASTING SOLAR FLARES
USING SHALLOW AND DEEP LEARNING TECHNIQUES

by

SUMI DEY

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Doctoral Program in Computational Science

THE UNIVERSITY OF TEXAS AT EL PASO

August 2022

Acknowledgements

I would like to express my sincere gratitude to my research advisor Dr. Olac Fuentes, faculty of the Department of Computer Science at The University of Texas at El Paso, for his advice, enduring patience and inspiration to complete my dissertation successfully.

I would also like to thank the other committee members Dr. Amy Wagler, faculty of the Department of Mathematical Science at The University of Texas at El Paso and Dr. Laura Boucheron, faculty of the Klipsch School of Electrical and Computer Engineering at The New Mexico State University for their effort and time.

I would like to thank the faculty and the staff of Computational Science Program of the University of Texas at El Paso for their hard work and dedication to complete my thesis.

My warmest gratitude goes to my parents and my sister for their love and constant support. They had always confidence in me even when I didn't have for myself.

My sincere thanks go to my all friends and loved ones for their contribution to my work.

Abstract

Solar flares, which are large sudden increases in X-ray flux, can damage satellite infrastructure, hamper power grids, disrupt Global Positioning Systems (GPS), and impair long-distance communication. Thus, the accurate prediction of solar flares has high practical importance and numerous approaches to the problem have been proposed. While these works have shown promising results, they have also highlighted the inherent difficulty of the problem, as solar flares are rare and heterogeneous events and the physical phenomena that give rise to them are still not well-understood. Solar flare prediction is normally posed as a classification problem, where a sequence of measurements is classified as a precursor or not of a solar flare within a given time frame. In this dissertation, we implemented and evaluated multiple algorithms for solar flare prediction posing the problem as a regression problem, focusing on the prediction of the maximum flux within a fixed future time interval. We compared conventional machine learning algorithms, state-of-the-art deep learning architectures tailored to time series regression, and a recently-proposed randomized shallow model called ROCKET, that has shown excellent results in other applications. Our experimental results show that ROCKET outperforms all other algorithms in terms of mean-squared error, while requiring much shorter training times than deep neural networks. Taking advantage of its efficiency, we propose an ensemble of ROCKET models, which leads to a further improvement in results.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Space weather refers to the time-varying conditions in the space surrounding the Earth due to solar activity. A variety of physical phenomena are associated with space weather, some of which have significant impacts on both human lives and technologies. Some examples of solar activities are solar flares, coronal mass ejections, high-speed solar wind, and solar energetic particles. The solar magnetic field is the ultimate source of all of these activities.

1.1 Solar Flares

1.1.1 What Are Solar Flares

A sudden release of distorted magnetic fields that produces a huge amount of energy and drives that energy into space creates a sudden flash of light known as a solar flare. Solar flares are considered as the largest explosive events in our solar system. Flares can last from minutes to hours. This electromagnetic emission travels at the speed of light, taking a little over eight minutes to reach the Earth. Sometimes the released energy accelerates very high energy particles such as protons and electrons, which take tens of minutes to reach the Earth [1].

1.1.2 Solar Flares Events

Solar flares are sometimes associated with other solar activities, for example, coronal mass ejections. This kind of events started to be recorded around 150 years ago. Some notable examples of solar flare events are following:

1. One of the most powerful solar flare events happened on September 1859. This event is known as the "Carrington event". The event was first reported by one of England's foremost solar astronomers named Richard Carrington. Because of that event, high energy particles entered into Earth's atmosphere and overpowered the Earth's protective magnetic field which resulted in a huge destruction on the ground. Telegraph systems were disrupted worldwide. Telegraph offices were ignited by the spark. Colorful aurora were seen at near tropical latitudes over Cuba, the Bahamas, Jamaica, El Salvador, and Hawaii. Telegraph workers felt the effects of the event in the following day also as the atmosphere was still strongly charged.

2. One of the historical powerful series of solar storms happened in August 1972. The solar storms were associated with solar flares, geomagnetic storms and high energy particles. This storm set off sea mines in Vietnam. This event disrupted satellites, hindered communication grids and electric service. This event occurred between the Apollo 16 and Apollo 17 lunar missions. If this would happen during one of the missions, the particles could have hit astronauts outside of Earth's protective magnetic field and the result could have been life-threatening.

3. From mid-October to early November 2003, one of the largest solar flare events occurred. This is actually the largest solar flare event that has been recorded by the Geostationary Operational Environmental Satellite (GOES) system. The Sun's magnetic field lines were stretched and then all of a sudden the magnetic field lines stretched beyond their limit. As a result, there was a gigantic explosion on the Sun's surface resulting in a coronal mass ejection (CME). CMEs are capable of exploding billions of tons of electrified gas and subatomic particles into space at a speed of five million miles per hour. This event disrupted communication satellite-based systems. A 90-minute blackout in Sweden was caused because of this event. Aircraft controllers had to change routes to avoid high altitudes near the polar regions.

1.2 Space Weather Monitoring

To predict the space weather, the Sun has been monitored continuously by satellites for several decades. Different satellites are the sources of data for different solar activities. The data are available in different forms, for example - images at multiple wavelengths, X-ray fluxes, proton fluxes, and electron fluxes. The followings are some examples of these satellites.

- SOHO - This is NASA/ESA's Solar and Heliospheric Observatory which gathers data on coronal mass ejections (CMEs).
- SDO - This is NASA's Solar Dynamics Observatory which is the source of data on sunspots, solar magnetic fields, and solar corona.
- GOES - The Geostationary Operational Environmental Satellite is operated by the National Oceanic and Atmospheric Administration (NOAA) to gather the data on X-rays and high-energy particles associated with solar flares, and high-energy electrons in geostationary orbit.
- HINODE - This is JAXA's solar physics satellite to gather X-ray image data on coronal holes, where high-speed solar wind originates.

1.3 Dissertation Goal

The goal of this dissertation is to develop techniques to forecast solar X-ray flux accurately from 1-minute X-ray flux time series data recorded by Geostationary Operational Environmental Satellite (GOES).

1.3.1 Significance of The Goal

Space weather events, in particular solar flares, can damage satellite infrastructure, hinder power grids, disrupt Global Positioning Systems (GPS) and damage long-distance com-

munication. A recent study [15] estimated that a single severe space weather event could cause an economic loss of between 0.5 trillion and 2.7 trillion USD. Airplane pilots, cabin crew and astronauts need to stay in space. They can be affected by the harmful radiation released from the particles. Forecasting X-ray flux is a necessary step towards fully automated prediction of space weather events.

1.3.2 Challenges

While several approaches to predict solar flares have been tried, the goal of fully automated mid-term and long-term prediction remains elusive. The main challenge for this task is that the type and amount of the data needed for accurate prediction is unknown. There are some other challenges as well.

- Since the signal being measured is weak, it presents low signal to noise ratios and quantization effects.
- Some operational failures, for example, solar eclipses or the replacement of the satellites, results in data loss.
- As old satellites are replaced, new sensors with different characteristics are used, reducing data consistency.
- Physical phenomena behind solar flare events are still not well understood by physicists and astronomers.

1.4 Research Questions

In this research, we are looking for the answers of the following questions:

1. How to forecast X-ray fluxes from time series data?
2. Which algorithm gives the most accurate result for this prediction task?

3. What wavelengths and temporal scales are most useful for the X-ray flux prediction?

1.5 Contributions

Our contributions are the following:

1. Presented the first systematic study of the prediction of solar X-rays that poses the problem as a regression problem.

2. Showed that complex neural models that excel at sequence prediction tasks are unable to exploit the information contained in large datasets of X-ray observations, being outperformed by simpler models, in particular the recently proposed ROCKET algorithm.

3. Proposed a faster modified implementation of the ROCKET algorithm.

4. Exploiting the efficiency of the modified implementation of the ROCKET, proposed an ensemble of models which provided the best results on the problem.

1.6 Outline

The remainder of this dissertation is organized as follows: relevant related work to this research is presented in Chapter 2. Chapter 3 describes the preparation of the dataset. Linear models are explored in Chapter 4. We describe ensemble models and presented the result of these models in Chapter 5. Deep learning models are discussed in Chapter 6. Later we explored and modified a shallow model and discussed it in Chapter 7. Finally, the conclusions and future works are presented in Chapter 8.

Chapter 2

Related Work

Space weather phenomena, and specifically solar flares, have been widely studied in recent years [38, 4, 19, 39, 15]. However, the mechanism of solar flare generation is still an unresolved mysterious topic in solar physics. Various kinds of works have been done to predict solar flares.

2.1 Space Weather Prediction Taxonomy

Forecasting space weather accurately is still in its infancy stage despite the availability of a huge amount of related data. Our proposed taxonomy, shown in Figure 2.1, divides space weather prediction techniques into two groups - *Data Driven* and *Model Driven*. Next we divide *Data Driven* into two groups, one consists of conventional machine learning algorithms, including *SVM, K-nearest neighbors* and *Logistic Regression* and the other is *Deep Learning*. Then Deep Learning is divided into two groups: *Convolutional Neural Networks* and *Recurrent Neural Networks*.

2.1.1 Model Driven Techniques

Techniques in this category do not rely on machine learning algorithms. This category includes statistical approaches to classify sunspots [30], predicting the probability for each solar active region to create solar flares in 24 hours [42], finding a statistical relationship between the solar magnetic field and flares [29], and predicting X-ray flares from active regions [5]. This also involves understanding the physical mechanisms of the origin of solar flares [41, 39, 21, 10, 20].

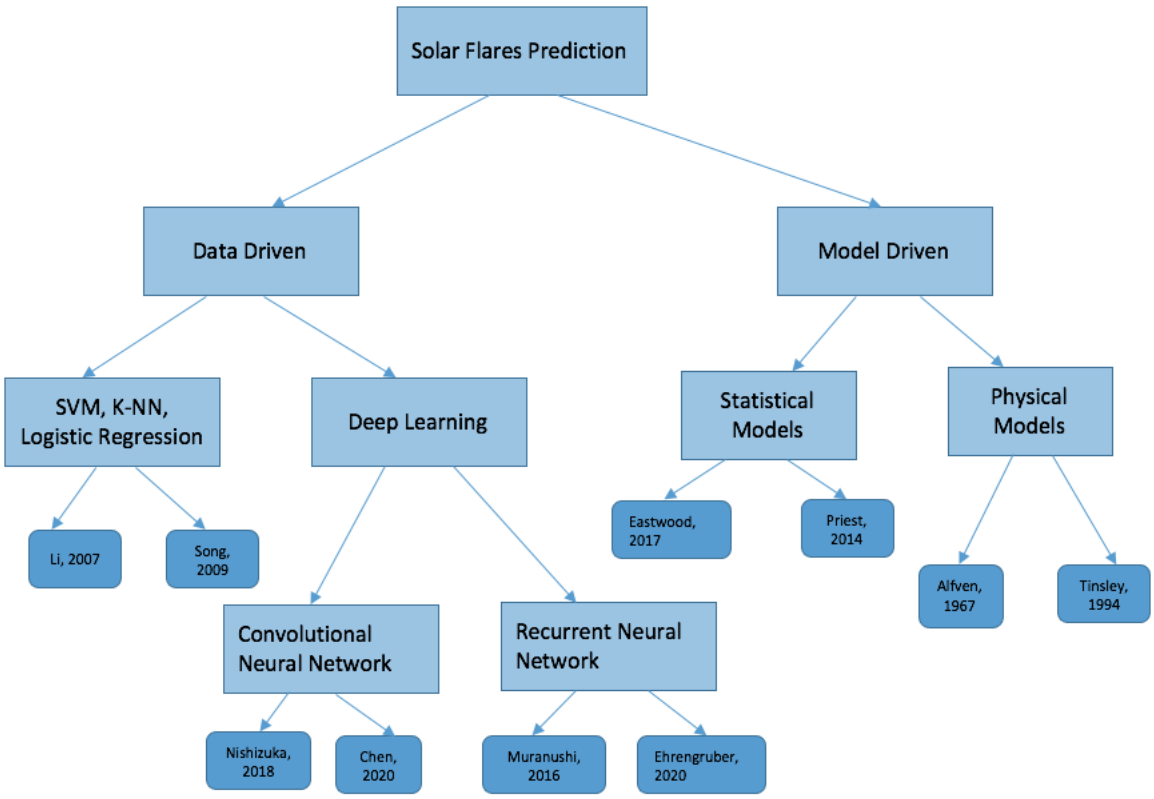


Figure 2.1: Solar flares prediction taxonomy

2.1.2 Data Driven Techniques

This techniques use machine learning algorithms to train models from data and then use those models to predict future events.

2.1.2.1 Traditional Machine Learning Techniques

These techniques involve machine learning approaches, for example, k-nearest neighbors, logistic regression, random forest, support vector machine, and shallow neural networks to exploit the huge amount of data available.

In [7] an approach using support vector regression to predict solar flare size and time-to-flare was presented. This technique used 38 features that describe the complexity of the magnetic field and are mapped into a continuous-valued label vector representing flare size or time-to-flare. [3] proposed a technique to select features for solar flare prediction. This study used 38 features that represent the spatial complexity of solar ARs and selected a subset of features within a sub-sampled classification based on histogram analysis. In [26, 25] solar flares and proton events are predicted applying a classification algorithm called SVM-KNN which is a combination of support vector machine and k-nearest neighbors. To predict the probability for each solar active region to produce X-, M-, or C- class flares during the next 1-day time period, a logistic regression model is proposed in [42]. [6] proposed a support vector machine model to forecast M- and X-class solar flares exploiting the Solar Dynamics Observatory's Helioseismic and Magnetic Imager data. In [49], a support vector machine is applied to do short-term solar power prediction from the National Solar Radiation Database (NSRDB). [40] established a correlation between solar flares and sunspot groups exploiting the National Geophysical Data Center. A combination of k-nearest neighbors and nearest centroid algorithms is implemented in [45] to predict solar flares based on the relation between the maximum ratio of the flare flux and non-flare background. In this work, the GOES X-ray flux data and the Space Weather Prediction Center flares catalog is used. Three machine learning techniques - k-NN, SVM and ERT -

are compared in [35] to forecast solar flares and the result showed that k-NN performs the best.

2.1.2.2 Deep Learning Techniques

This subgroup of techniques use deep neural networks for the prediction. In recent years, deep neural networks have lead to breakthrough results on various types of problems. We divided the deep neural networks into two categories - 1) Convolutional Neural Networks and 2) Recurrent Neural Networks.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have achieved great success in face recognition [23], handwritten character recognition [24], image classification [13], image captioning [22], and visual game playing [31], among many others.

In [37], a deep convolutional neural network is applied to forecast solar flares within 12 hours using SOHO and SDO full-disk magnetograms. [27] used HMI and SDO magnatograms of solar active regions to predict solar flares in 24 hours using a CNN. [2] proposed a two-stage deep learning model to forecast solar flares. In the first stage, sunspot groups on SDO/HMI images are detected using a CNN model and then these images are fed into another CNN model to forecast the solar flares. [32] developed a deep Convolutional Neural Network to predict solar flares in a time window from 20 minutes to 120 minutes using Geostationary Operational Environmental Satellite (GOES) X-ray flux data. [34] presented a deep neural network named Deep Flare Net (DeFN) to calculate the probability of flares occurring in the next 24 hours using SDO images.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have had great success for recognizing patterns that are defined by temporal distance. Recurrent Neural Networks have been applied in music

generation [16, 8], text generation [43, 17], image generation [18], and image caption generation [47, 22], among others.

In [14], an RNN is used to forecast sunspot numbers, radio solar flux and mean solar magnetic field values. [44] predicts the maximum solar flare class produced by an active region from the SDO/HMI vector magnetic field using LSTM. To predict geomagnetic storms from solar wind data, a recurrent neural network is used in [46]. Prediction of two sunspot-related time series was done in [28] using a recurrent neural network. Weather conditions and isolation have influence on photovoltaic (PV) systems, accepted as an alternative source of energy. In [48] a recurrent neural network is used to forecast the power output of the PV system. To forecast hourly and daily solar irradiation, a diagonal recurrent wavelet neural network is used in [9]. Solar wind prediction is made in [33] using a hybrid computation method combining wavelet decomposition and recurrent neural networks.

2.2 Summary of Existing Approaches

Most work aimed at forecasting solar flares is posed as a classification problem, where a sequence of measurements is classified as a precursor or not of a solar flare within a given time frame. Fluxes above 10^{-5} watts per square meter were considered positive instances (corresponding to X-class and M-class flares, the two strongest types) while the remaining fluxes are regarded as negative instances. Consequently, the binary labeling results in an unbalanced classification problem, as solar flares are relatively rare, with 50 X-class and 742 M-class flares since 2001. In addition, while binarization may help reduce the effects of noise and outliers, it also results in information loss, which may be especially harmful when predicting rare events. Moreover, since flares are heterogeneous and have no clear identified precursors, machine learning models have had only moderate success in this prediction task, performing only slightly better than probabilistic random guessing.

Previous works have shown promising results because of the availability of huge amount

of data, different types of data and better machine learning algorithms. However, the existing results are still not satisfactory enough to apply in real world scenarios.

Chapter 3

Dataset

3.1 Data Collection

In this thesis, we used data collected by GOES-8 through GOES-15. Figure **??** presents a list of the satellites in the GOES series and their time spans. These satellites have monitored solar X-ray fluxes, proton fluxes, and electron fluxes for several decades now. As our goal is to predict solar X-ray fluxes, we created our own dataset using multiple queries to the mirror server (http://darts.isas.ac.jp/pub/solar/sswdb/goes/xray/).

3.2 Data Description

The data are available with two different intervals - 1-minute and 5-minutes. We chose to exploit the 1-minute data because it has 5 times more information about the solar fluxes.

There are two wavebands for the X-ray channels:

- Soft X-ray channel with wavelength of 0.1 to 0.8 nanometers.
- Hard X-ray channel with wavelength of 0.05 to 0.4 nanometers.

We chose the soft X-ray channel for this thesis. We collected the data from 1998 through 2020. There are more than 10 million observations within the time range. Within these 10 million observations, there are less than 1% flare samples.

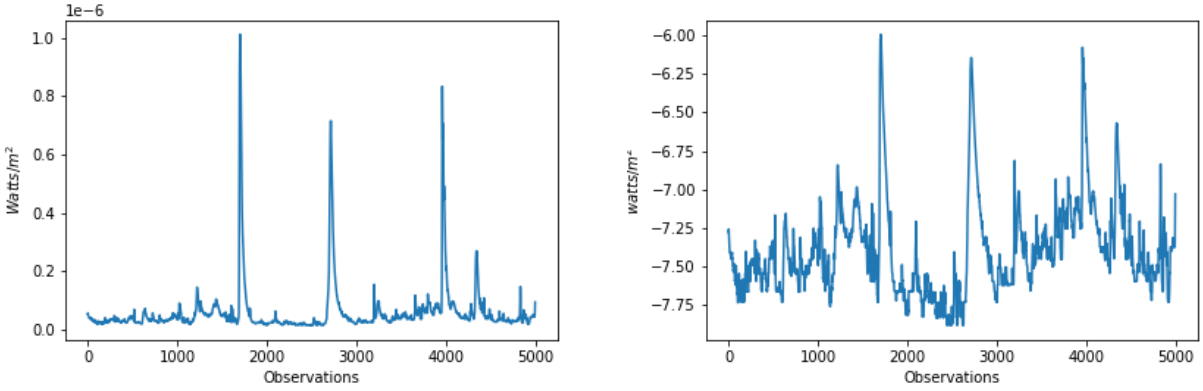
3.3 Data Preparation

X-ray flux, measured in watts per square meter, have very small magnitudes. The range for these magnitudes is $[10^{-9}, 10^{-3}]$. This is why we rescaled the X-ray flux into logarithmic scale. To evaluate machine learning algorithm, we split the dataset into a training and a test set as follows. The whole dataset was divided into chunks. Each of the chunks has one week information. From every chunk, the first 80% of the data are assigned to the training set and last 20% are assigned to the test set. Table 3.1 shows some samples before and after scaling.

Table 3.1: GOES list

GOES	Time Span
GOES-1	1975-1985
GOES-2	1977-1979
GOES-3	1978-1987
GOES-4	1980-1982
GOES-5	1981-1984
GOES-6	1983-1989
GOES-7	1987-1996
GOES-8	1994-2004
GOES-9	1995-2005
GOES-10	1997-2009
GOES-11	2000-2011
GOES-12	2001-2013
GOES-13	2006-present
GOES-14	2009-present
GOES-15	2010-present

In the experiments, features are a sequence of fluxes in the lookback and target is the



(a) X-ray flux in original scale

(b) X-ray flux in logarithmic scale

Figure 3.1: X-ray flux in different scales

maximum in the prediction window. For the length of prediction window L, we explored the length of lookback $L, 2L, 3L$. After analyzing the results with different length of lookback, we decided to use $2L$ as the length of lookback for the length of prediction window L. In the experiments, we also analyze the time to train a model which we will address as execution time.

Chapter 4

Linear Models

This chapter focuses on different linear models and their performance to predict solar X-ray fluxes. The reasons behind exploring linear models are:

- Linear models are simple and numerous implementations are readily available.
- Learned parameters are easy to interpret.
- The quality of their results is often competitive with that of more complex algorithms at a fraction of the computational cost.

No existing approach considers the solar flare prediction task as a regression problem. So we don't have any previous results that we can compare our work with. That is why, as a starting point, we have considered three simple non-machine learning baselines to compare the performance of linear models.

- **Baseline-1:** The prediction is the average flux in the lookback window.
- **Baseline-2:** The prediction is the average flux in the second (most recent) half of the lookback window.
- **Baseline-3:** The prediction is the last observed flux in the lookback window.

The performance metric used for all experiments in this thesis is the mean squared error, given by:

$$MSE = (1/n)\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

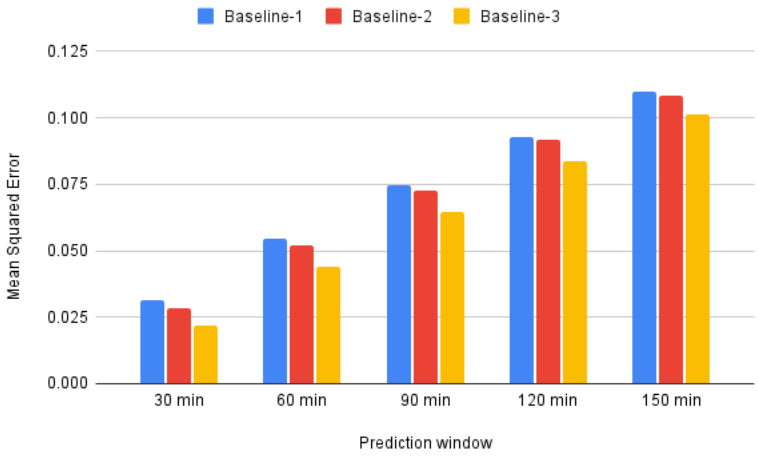


Figure 4.1: Graphical representation of the performance of all non-machine learning baselines

where n is total number of samples, $\mathbf{Y}$ is the vector of actual target values and $\hat{\mathbf{Y}}$ is the vector of predicted values.

The performance of these non-machine learning baselines is shown in Table 4.1 and for visualization purposes, the results are also shown in Figure 4.1

Table 4.1: Performance of non-machine learning baselines in forecasting fluxes

Prediction Window	**Baseline-1**	**Baseline-2**	**Baseline-3**
30 min	0.03141	0.02816	0.02167
60 min	0.05454	0.05219	0.04376
90 min	0.07463	0.07284	0.06460
120 min	0.09302	0.09159	0.08373
150 min	0.10973	0.10845	0.10118

From Figure 4.1 it is clear that **Baseline-3** is the best baseline. So we will compare

the performance of linear models with this baseline.

4.1 Linear Regression

Linear Regression is the simplest statistical model that is widely used for prediction tasks. The mathematical formula for this model is:

$$\hat{\mathbf{Y}} = \mathbf{wX} + \mathbf{b}$$

where $\mathbf{X}$ is the feature vector, $\mathbf{w}$ is the corresponding weight vector, $\mathbf{b}$ is the scalar bias and $\hat{\mathbf{Y}}$ is the target variable.

The cost function of this model is the sum of squared errors of the predicted outcome as compared to the actual outcome. This can be expressed mathematically as:

$$Cost(W) = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} w_j x_{ij} \right)^2$$

where i represents the sample number and j represents the feature number.[1]

This model uses gradient descent to minimize the cost function. The gradient is the partial derivative of the cost function with respect to the weights. To minimize the cost, this technique iteratively updates the weights using the gradient. The updating step is given by:

$$w_j^{t+1} = w_j^t + 2\eta \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k=1}^{m} w_k x_{ik} \right)$$

If the gradient is small enough, this means we are close to the optimal solution and updating the weights won't have a substantial impact on the prediction.

[1] For the simplicity, bias term is ignored here.

4.1.1 Data preparation

For the experiments, features are extracted from the history of fluxes withing a particular time interval, called lookback. In this work, the target is the maximum flux in a future interval with a specific length, which is called the prediction window. In particular, for all our experiments, the length of the lookback is twice the length of the prediction window and there is no delay from the end of the lookback to the beginning of the prediction window. For example, if the features are the fluxes from 12:00 AM to 1:59 AM, then the target will be the maximum flux from 2:00 AM to 2:59 AM.

4.1.2 Experimental results

We used the well-known sklearn library for these experiments. Table 4.2 shows the performance of the Linear Regression model to predict the solar fluxes and the result is compared the performance with the Baseline-3. The graphical result is shown in Figure 4.2.

Table 4.2: Performance of Linear Regression in forecasting fluxes

Prediction Window	**Baseline-3 (Train MSE)**	**Baseline-3 (Test MSE)**	**Linear Regression (Train MSE)**	**Linear Regression (Test MSE)**	**Improvement Percentage on Test Set**
30 min	0.02156	0.02167	0.01898	0.01897	12.46%
60 min	0.04359	0.04376	0.03617	0.03589	17.98%
90 min	0.06401	0.06460	0.0506	0.05018	22.32%
120 min	0.08315	0.08373	0.0631	0.06216	25.76%
150 min	0.1032	0.10118	0.07427	0.07225	28.59%

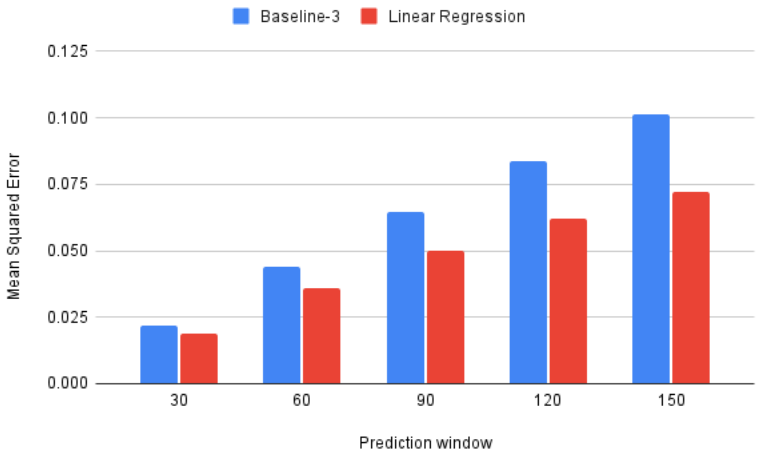


Figure 4.2: Graphical representation of the performance of Linear Regression

From the Figure 4.2, we can see that Linear Regression outperforms the baselines. We can also see that the performance of Linear Regression improves 12% over the baseline for 30-minutes prediction window. As the problem becomes more complex meaning as the prediction window gets longer, Linear Regression improves over the baseline more than 12%. For the maximum prediction window which is 150-minutes in our experiments, Linear Regression improves over the baseline by more than 29%.

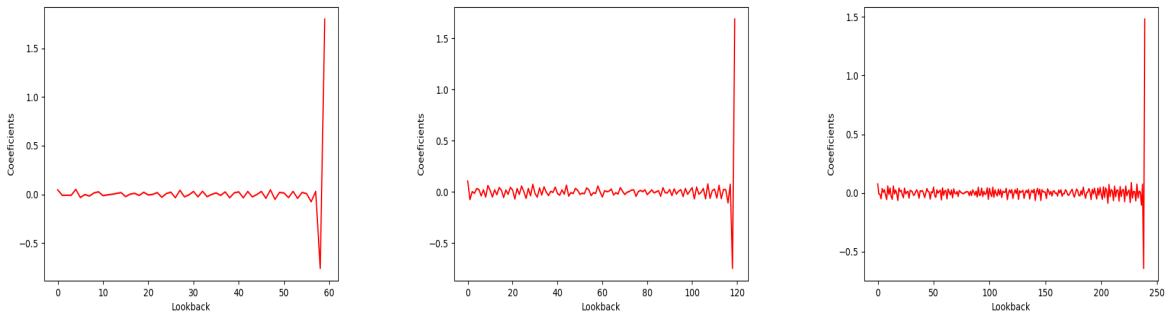


Figure 4.3: Feature coefficients for Linear Regression for lookbacks of 60, 120, and 240 minutes.

Figure 4.3 shows the weights that are assigned by linear regression to predict the max-

imum flux in different prediction windows. From Figure 4.3, we can see that the most important feature for this prediction task is the last feature which is the latest flux in the lookback and the second most important feature is the next-to-last flux. The coefficients for the remaining features are small, very close to 0.

In Baseline-3, we have considered the latest flux in the lookback as the prediction. From the result of coefficient analysis of linear regression, we found that this model also suggests that the latest flux in the lookback is the most important feature.

4.2 Ridge Regression

From the result of linear regression, we have seen that linear regression assigns higher weight to the latest fluxes. We know that allowing coefficient values to increase without bound can lead to overfitting or to situations where a single feature dominates the predictions, making learning from other potentially useful features impossible. For this reason, we decided to explore whether some restrictions on the weights can improve the prediction task.

Ridge Regression is actually a kind of Linear Regression with more restrictions on the weights. This cost function in the model penalizes weights that are too large. The cost function for this model can be expressed as:

$$(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} \left(Y_i - \sum_{j=1}^{m} (W_j X_{ij})^2 \right) + \lambda \sum_{j=i}^{m} W_j^2$$

If $\lambda \to 0$, then this is equivalent to the cost function of Linear Regression. Ridge Regression also uses gradient descent to minimize the cost function, meaning it updates the weights using the gradient. The updating step can be expressed as

$$w_j^{t+1} = (1 - 2\lambda\eta) w_j^t + 2\eta \sum_{i=1}^{n} x_{ij} \left(y_i - \sum_{k=1}^{m} w_k x_{ik} \right)$$

where η is the learning rate. From the above equation, we can see that the second term in the right hand side is the same as updating the weights of Linear Regression. The only difference in Ridge Regression is this technique reduces the weights by a factor first and then follows the same update rule as Linear Regression. By restricting the weights, this model also often converges faster than Linear Regression.

4.2.1 Experimental results

We show the performance of Ridge Regression in Table 4.3 and the result is compared with the non-machine learning baseline. Figure 4.4 represents the result in Table 4.3 in graph.

Table 4.3: Performance of Ridge Regression in forecasting fluxes

Prediction Window	**Baseline-3 (Train MSE)**	**Baseline-3 (Test MSE)**	**Ridge Regression (Train MSE)**	**Ridge Regression (Test MSE)**	**Improvement Percentage on Test Set**
30 min	0.02156	0.02167	0.01898	0.01897	12.46%
60 min	0.04359	0.04376	0.03617	0.03589	17.98%
90 min	0.06401	0.06460	0.05058	0.05018	22.32%
120 min	0.08315	0.08373	0.06308	0.06214	25.79%
150 min	0.10132	0.10118	0.07427	0.07223	28.61%

The result shows that the performance of Ridge Regression is the same as that of Linear Regression. We have also explored the performance of Ridge Regression for different values of λ, namely $0.0001, 0.001, 0.01, 0.1, 1$, and 10, observing very little variation. Table 4.3 shows that the error in the training set is slightly bigger than the error in the test set, thus there in no overfitting of the training data.

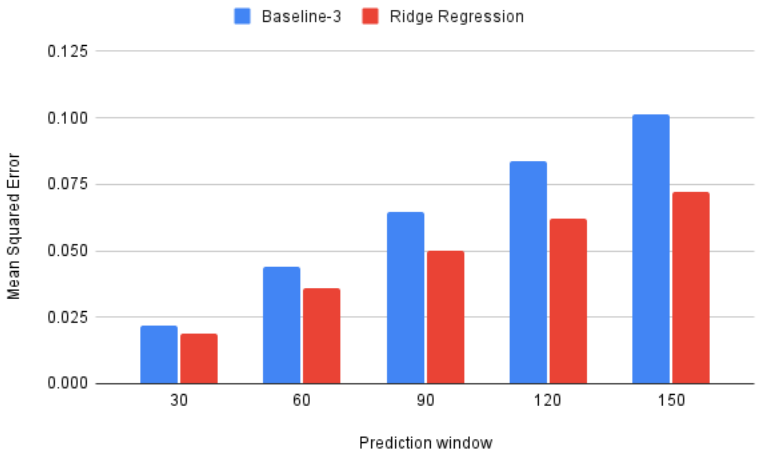


Figure 4.4: Graphical representation of the performance of Ridge Regression

Figure 4.5 shows the weights that are assigned by Ridge Regression to predict the maximum flux in different prediction windows. From Figure 4.5, we can see that the most important feature for this prediction task is the last feature which is the latest flux in the lookback like Linear Regression. The second most important feature is the second last flux. As in Linear Regression, the remaining features are close to 0.

4.3 Lasso Regression

We wanted to explore important features for this prediction task. Lasso Regression is used to find the important features for prediction tasks. The cost function for Lasso Regression is

$$(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} \left(Y_i - \sum_{j=1}^{m} (w_j X_{ij})^2 \right) + \lambda \sum_{j=i}^{m} |W_j|$$

If $\lambda \to 0$, then this is equivalent to the cost function of Linear Regression. Instead of gradient descent, this model uses a different technique to minimize the cost function, which is called coordinate descent. This optimization technique excludes one feature at a time and

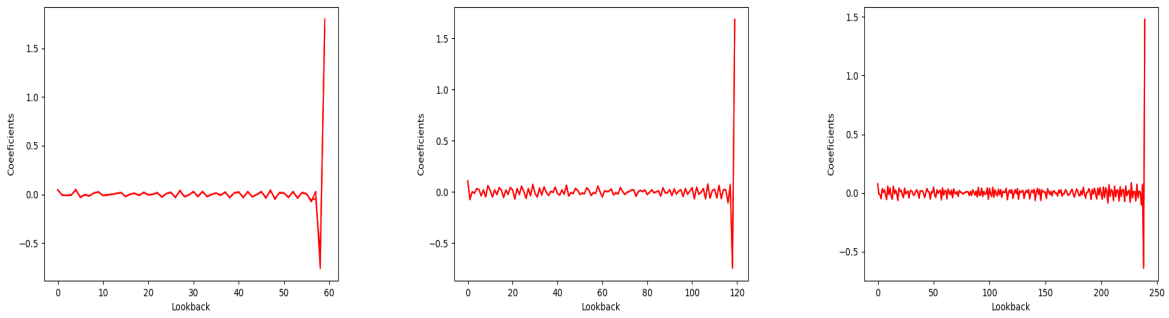


Figure 4.5: Feature coefficients for Ridge Regression for lookbacks of 60, 120, and 240 minutes.

checks the difference between the actual value and the prediction. If the difference is small enough, then the model assigns 0 weight to the corresponding feature as the model can predict well enough without that feature. Otherwise, it reduces the weights by a constant factor. This is how this model removes some of the features, which leads to feature selection.

The difference of the cost function between Ridge Regression and Lasso Regression is that Lasso Regression penalizes the magnitude of the weight instead of the squared value of the weight.

4.3.1 Experimental results

The numerical results for Lasso Regression are presented in Table 4.4 and the graphical results are shown in Figure 4.6.

Results show that the performance of Lasso is very close to Linear and Ridge Regression. As the prediction window size increases, the performance of Lasso Regression relative to the baseline also gets better. For the maximum prediction window (in our experiments), 150-minutes, Lasso Regression improves 28% over the baseline. We have also explored the performance of Lasso Regression for different values of $\lambda = 0.0001, 0.001, 0.01, 0.1, 1$ and

Table 4.4: Performance of Lasso Regression in forecasting fluxes

Prediction Window	**Baseline-3 (Train MSE)**	**Baseline-3 (Test MSE)**	**Lasso Regression (Train MSE)**	**Lasso Regression (Test MSE)**	**Improvement Percentage on Test Set**
30 min	0.02156	0.02167	0.01908	0.01937	10.61%
60 min	0.04359	0.04376	0.03628	0.03624	17.18%
90 min	0.06401	0.06460	0.05068	0.05049	21.84%
120 min	0.08315	0.08373	0.06318	0.06241	25.46%
150 min	0.10132	0.10118	0.07437	0.07247	28.38%

10. The best result we found was for $\lambda = 0.0001$.

Figure 4.7 presents the weights that are assigned by Lasso Regression to predict the maximum flux in different prediction window. From Figure 4.7, we can see that the most important feature for this prediction task is the last feature which is the latest flux in the lookback like Linear Regression. The second most important feature is the next-to-last flux, as in the other regression models. Also, the coefficients for the remaining features are close to 0.

For better visualization, the performances of all the linear models are shown in Figure 4.8 and from the Figure 4.8, we can see that all linear models outperform the baseline and have similar performances, with Linear Regression being the best among them by a small margin.

Figure 4.9 presents the execution time for all the linear models. From Figure 4.9, we can tell that Ridge Regression is the most time efficient linear model and it's because it

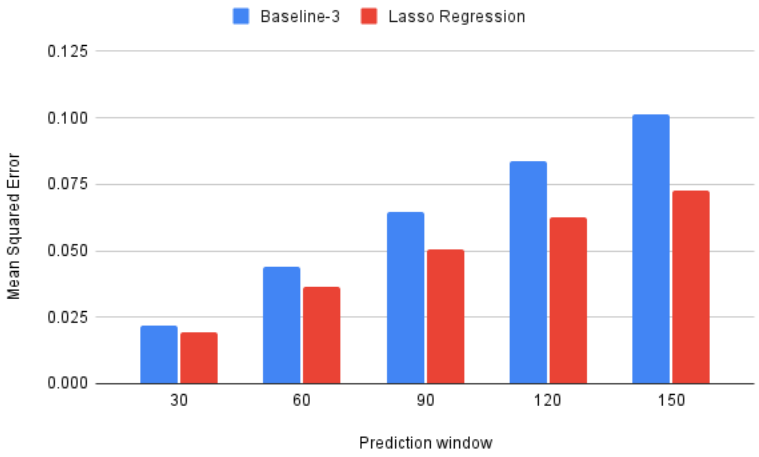


Figure 4.6: Graphical representation of the performance of Lasso Regression

penalizes the weights and that penalization strategy helps the model converge faster than Linear Regression.

4.4 Summary

From the results of the linear models, we make the following observations:

- As expected, linear models outperform the baseline.
- As the prediction window grows, the relative improvement provided by linear models with respect to the baseline grows as well. For linear regression, the improvement increases from 12% for a 30-minute prediction window to 29% for a 150-minute prediction window.
- While Ridge and Lasso Regression converge slightly faster than Linear Regression, their performance is nearly identical.
- The performance of all three linear models on training data is very similar to their performance on test data, due to the fact that these simple models do not have enough parameters to overfit the data.

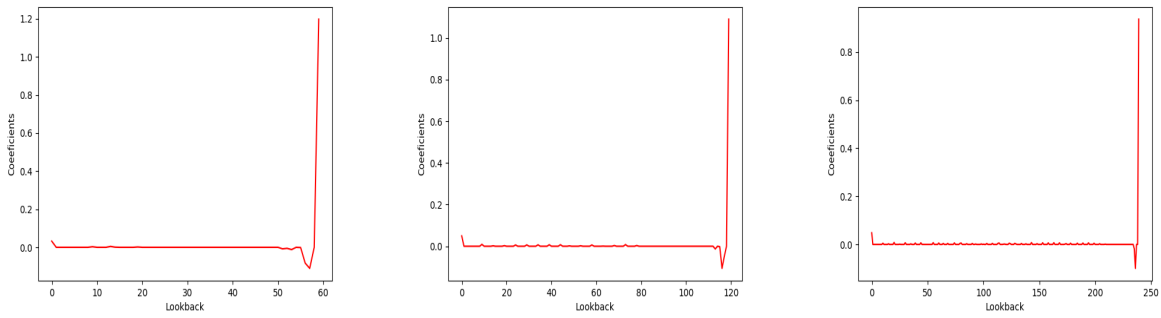


Figure 4.7: Feature coefficients for Lasso Regression for lookbacks of 60, 120, and 240 minutes.

- Results for Ridge and Lasso Regression are not sensitive to the choice of λ within a wide range of values (we tried 0.0001 to 10). The smallest λ value, which makes Ridge and Lasso Regression essentially equivalent to Linear Regression, yielded the lowest MSE, by a small margin.

- Coefficient analysis (Figure 4.3, Figure 4.5, Figure 4.7) shows that the latest few fluxes in the lookback are the most important features. This also suggests that linear models are unable to extract meaningful patterns from the whole lookback window.

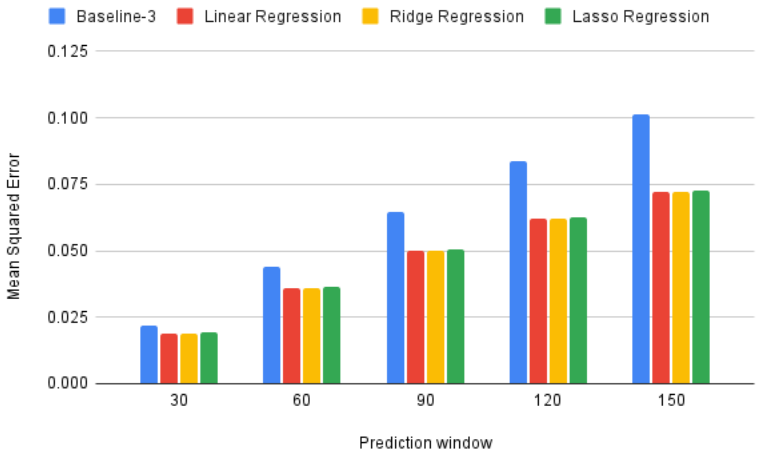


Figure 4.8: Graphical representation of the performance of all linear models

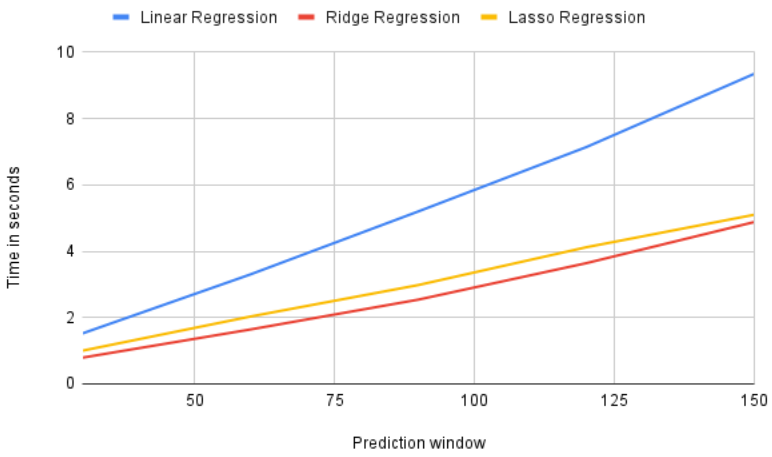


Figure 4.9: Execution time for all linear models

Chapter 5

Ensemble Models

Ensemble models are very popular for regression and classification tasks. They work by combining the results of several basic models, most commonly decision or regression trees. Ensembles usually perform better than any of the individual models that makes them up, provided the individual models are better than random guessing and that their errors are not strongly correlated. In this chapter we analyze the performance of tree-based ensemble models for X-ray prediction. Trees are based on a hierarchical structure where each internal node represents a test comparing one of the features of the data with a threshold. Based on that test, the tree branches out in two different directions. This process is repeated, creating a binary tree that splits the original data recursively in two sections until a termination condition is reached. Common termination conditions include a goal variance, a maximum depth allowed, or a minimum number of training examples associated with the sub-tree. At that point we have reached a leaf of the tree, which contains the predicted regression value.

5.1 Random Forests

Random forests combine several decision trees and these trees are different from each other. Each tree is trained on a random selection of training examples, sampled with replacement, and a random subset of the features. Each tree is built independently, so this task can be parallelized. Prediction is done by calculating the prediction for every tree in the forest and averaging all the results to obtain the final prediction.

5.1.1 Experimental results

Data are prepared in the same way as for linear models (*section 4.1.1*). We again used the sklearn library for our experiments. In our experiments, we used different numbers of individual trees (parameter *n_estimators* in sklearn) - 30, 50, and 70 and maximum depth allowed (parameter *max_depth* in sklearn) - 5, 10, and 15. The best result found was for *n_estimators* = 70 and *max_depth* = 15. Table 5.1 presents the performance of Random Forests numerically. A comparison of the Random Forest algorithm and Linear Regression is shown in Figure 5.1. We are comparing the performance of Random Forests with Linear Regression only because we have seen in the previous chapter that Linear Regression gives the best results among the linear models, thus it will be considered as the new baseline.

Table 5.1: Performance of Random Forests Regression in forecasting fluxes

Prediction Window	**Linear Regression (Train MSE)**	**Linear Regression (Test MSE)**	**Random Forests Regression (Train MSE)**	**Random Forests Regression (Test MSE)**	**Improvement Percentage on Test Set**
30 min	0.01898	0.01897	0.01190	0.01795	5.38%
60 min	0.03617	0.03589	0.02246	0.03305	7.91%
90 min	0.05058	0.05018	0.03095	0.04503	10.26%
120 min	0.06308	0.06216	0.03779	0.05449	12.34%
150 min	0.07427	0.07225	0.04353	0.06255	13.43%

Figure 5.1 shows that the performance of Random Forests improves 5% on the 30-minutes prediction window over Linear Regression. This is because random forests are more complex model than Linear Regression and this increased complexity helps the model to learn non-linear models. Moreover, the relative performance improvement of Random

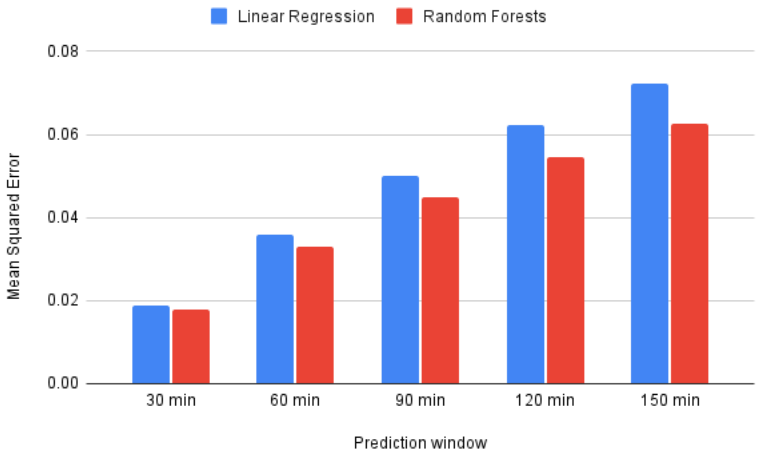


Figure 5.1: Graphical representation of the performance of Random Forests Regression

Forest increases as the prediction window becomes larger. In our experiments, we used 150-minutes as the maximum prediction window and for this experiments Random Forests improves over Linear Regression by over 13%.

5.2 XGBoost

XGBoost, which stands for Extreme Gradient Boosting, is another ensemble model which is based on a collection of decision trees. Each tree is trained on a different subset of samples and these trees are built sequentially such that each subsequent tree learns from the previous tree and reduces the error in the previous tree.

XGBoost follows these steps:

- An initial tree M_0 is trained on a random subset of samples to predict the target y.
- The residuals (difference between the target and the prediction made by tree M_0) is used to create a regression tree h_1 that computes the mean of the residuals at each leaf of the tree.

- Next tree M_1 will be trained on a different random subset (without replacement) and add 50% of the samples from the previous tree with larger residual errors and for this 50% samples target will be the sum of the prediction made by previous tree M_0 and h_1.

5.2.1 Experimental results

In our experiments, we used different values of *n_estimators* - 30, 50, 70, and 100 and *max_depth* - 5, 10, and 15. The best result found was for *n_estimators* = 70 and *max_depth* = 10. The numerical result is shown in Table 5.2. The performance of XGBoost is compared with Linear Regression and the result is presented in Figure 5.2.

Table 5.2: Performance of Random Forests Regression in forecasting fluxes

Prediction Window	**Linear Regression (Train MSE)**	**Linear Regression (Test MSE)**	**XGBoost Regression (Train MSE)**	**XGBoost Regression (Test MSE)**	**Improvement Percentage on Test Set**
30 min	0.01898	0.01897	0.01360	0.01790	5.64%
60 min	0.03617	0.03589	0.02481	0.03295	8.19%
90 min	0.05058	0.05018	0.03320	0.04479	10.74%
120 min	0.06308	0.06216	0.03959	0.05443	12.44%
150 min	0.07427	0.07225	0.04488	0.06186	14.38%

Figure 5.2 tells that XGBoost performs 5% better than Linear Regression on a 30-minutes prediction window, which is slightly better than Random Forest. XGBoost also improves relative to Linear Regression as the prediction window gets bigger. This algorithm improves 14% over Linear Regression on a 150-minute prediction window. This result is 1% better than the result of Random Forests on 150-minutes prediction window. The reason

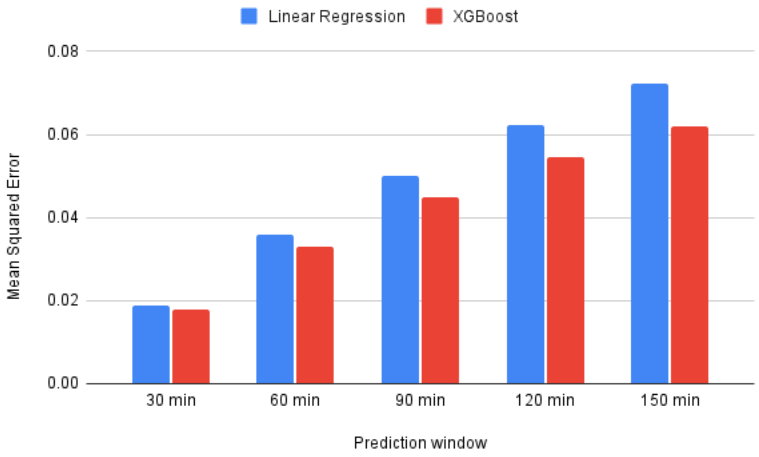


Figure 5.2: Graphical representation of the performance of XGBoost Regression

is that each tree is built sequentially to reduce the error in the previous tree. That helps to obtain a better result.

The comparison of both ensemble techniques is presented in Figure 5.3 and the execution time for each ensemble technique is shown in Figure 5.4.

Figure 5.3 shows that the performance of XGBoost is slightly better than Random Forests. This happens because in Random Forests, all the trees are built independently and at the same time. So the result is not shared among the trees. But in XGBoost, as the trees are built sequentially and the result of the current tree is shared with the next tree in the aim to reduce the error in the next tree. So XGBoost has the advantage to learn from the mistakes made by previous trees.

Figure 5.4 shows that Linear Regression is the most time-efficient algorithm. It's because it is one of the simplest models. XGBoost takes more time than Random Forests because the trees in XGBoost are built sequentially while the trees in Random Forests can be built in parallel.

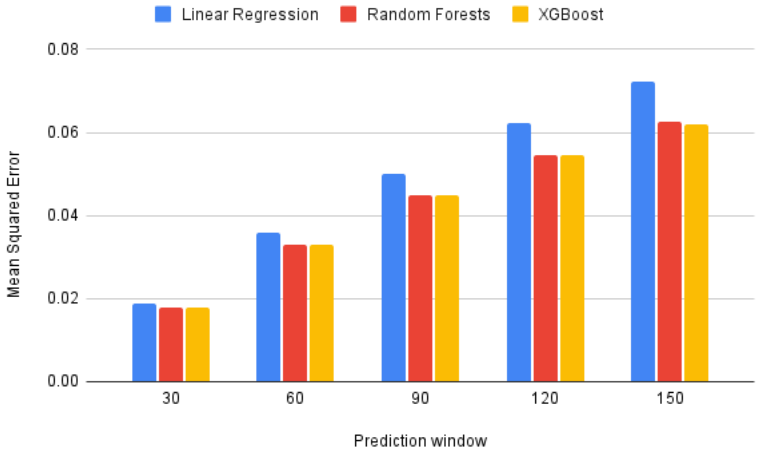


Figure 5.3: Comparison of the performance of all ensemble models

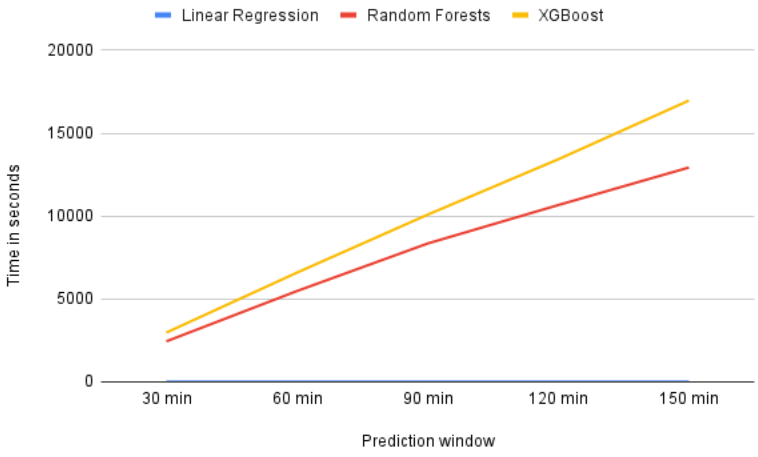


Figure 5.4: Execution time for all ensemble models

5.3 Summary

From the results on ensemble techniques, we can conclude the following:

- Ensemble models improve over Linear Regression by 14%. This is likely due to their increased complexity, which allows them to learn highly non-linear models.
- The performance of ensemble models improves more as the prediction window grows. XGBoost improves 5% over Linear Regression for a 30-minute prediction window and for a 150-minute prediction window, the result improves by 14%.
- The performance of XGBoost is 1% better than the performance of Random Forests, showing that residual learning provides a small advantage in this problem.
- Random Forests is time efficient because the trees can be built independently and simultaneously.

Chapter 6

Neural Networks

Neural Networks have had huge success in solving problems in different domains, for example, image classification, text classification, speech recognition, video captioning and many others. This is the motivation to explore these models to forecast solar X-ray flux. In this chapter, we are going to discuss several neural models. These neural models include:

- 1-D CNN (One-dimensional Convolutional Neural Network)
- LSTM (Long-short Term Memory)
- N-BEATS (Neural basis expansion analysis for interpretable time series) - the state-of-the-art technique to forecast the time series
- 1D CNN with Residual Learning

We developed different neural networks using thr Keras library. Initially we developed a 1D CNN architecture with more than 1 million parameters, and an LSTM architecture with more than 70 thousand parameters. But we found that there were no significant differences in performance between a smaller and a bigger architectures. That is why we ended up exploring the performance of the deep models with small architectures only, as bigger architectures are computationally expensive and offered no performance improvements.

We also analyze the performance of these models and compare the result with our new baselines - Linear Regression and the best ensemble model, XGBoost.

6.1 1D CNN

1-D Convolutional Neural Networks usually perform well in sequence processing. A 1D CNN considers time as the independent dimension and extracts interesting features from local input patches. 1-D CNNs have been successfully used in many fields, including audio classification, machine translation, and many others.

6.1.1 Data Preparation

Our implementation of 1D CNN requires the input to be shaped as a 3D tensor - (number of samples, lookback, number of features), where lookback is the number of previous samples we are looking at for each sample. So we need to convert our input into a 3D tensor to feed into the 1D CNN. We can customize the lookback to analyze the behavior of the network.

6.1.2 Model Architecture

We used 16 filters of size 5 in the first layer and a Rectified Linear Unit as the activation function. The output of the first layer of the network is fed into the second layer as input. The second layer is identical to the firat one, with 16 kernels of size 5 and ReLU activation. After using stack of convolutional layers we used a maxpooling layer to reduce the complexity of the output and to prevent overfitting the data. The output of this maxpooling layer will be half of the input of the layer as we have used maxpooling size 2 here. On the top of the maxpooling layer, we used a flatten layer to convert the output of the maxpooling layer, which is a 2 dimensional arrays, into a single linear vector. On top of that we have used a fully connected layer with 250 units and linear activation. Mean squared error is the loss function.

6.1.3 Prediction

While training, the network learns all the parameters from the training data. Then these learned parameters are used to do the prediction.

6.1.4 Experimental results

Table 6.2 presents the performance in numerical results of 1D CNN compared to the best linear model - Linear Regression and the best ensemble model - XGBoost, while Figure 6.1 shows the graphical representation of the Table 6.2.

Table 6.1: Performance of 1D CNN vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**1D CNN (Train MSE)**
30 min	0.01898	0.01360	0.02760
60 min	0.03617	0.02481	0.04650
90 min	0.05058	0.03320	0.06240
120 min	0.06308	0.03959	0.07270
150 min	0.07427	0.04488	0.08190

The result shows that the performance of 1D CNN is close to Linear Regression when the prediction window is less than or equal to 2 hours and the performance starts to degrade when the prediction window is more than 2 hours. From the poor performance of 1D CNN we can infer that as the solar flare events are very rare, that makes it difficult for 1D CNN to learn the hidden pattern to predict accurately.

Table 6.2: Performance of 1D CNN vs Linear Regression vs XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**1D CNN (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.02710	None
60 min	0.03589	0.03295	0.03707	None
90 min	0.05018	0.04479	0.05216	None
120 min	0.06216	0.05443	0.06226	None
150 min	0.07225	0.06186	0.08076	None

6.2 LSTM

LSTM is very popular in time series prediction because it can capture long-term dependencies in the data, which can be used to predict the future trend in the time series. This has a lot of flexibility in modeling a problem. For instance - when we are required to predict the current timestep using the previous information, we can use many to one model. Sometimes we also need to predict for multiple timesteps in the future at once using the same previous information, in that case we can use many to many model. For the solar flux prediction task, we used a many to one model as we want to predict the maximum flux in the prediction window only from a history of fluxes.

6.2.1 Data Preparation

LSTM also requires a 3D tensor as input - (number of samples, number of features, lookback). So to be fed into the network the data are converted into three dimensional data. The behavior of the network is analyzed for different values of lookback.

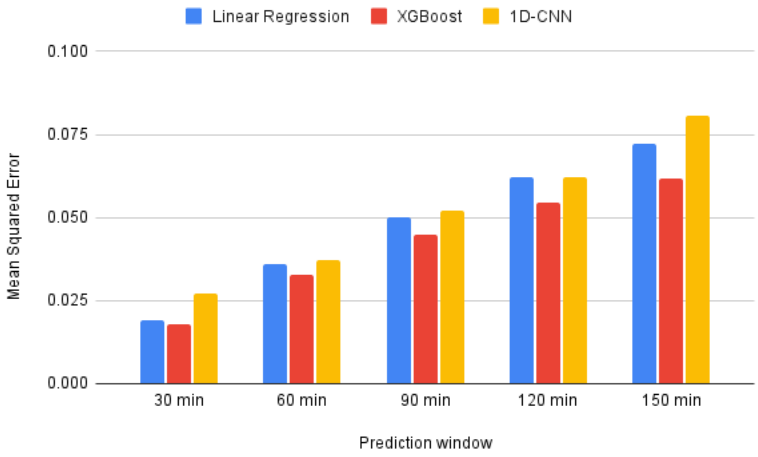


Figure 6.1: Comparison of the performance of 1D CNN with Linear Regression and XGBoost

6.2.2 Model Architecture and Prediction

We used a single LSTM layer with 16 units and hyperbolic tangent activation function. Then we used a dropout layer with 20% rate to randomly set the result from the LSTM layer to 0. Then this was fed into a fully connected layer with 64 units. The activation function for the fully connected layer is a linear function. On top of that we used the output layer. Mean squared error is the loss function we used here. Prediction is done in the same way as 1D CNN. The performance metric is also the same.

6.2.3 Experimental results

The performance of LSTM is shown in Table 6.4 and the numeric result is compared with the best baseline, Linear Regression and XGBoost. For visualization purpose, the result described in Table 6.4 is presented in Figure 6.2.

The result shows that the performance LSTM is very poor. It can't even outperform the original baseline. This poor performance suggests that LSTM might not be able to learn the pattern in the time series as there is very limited information about the solar flare

Table 6.3: Performance of LSTM vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**LSTM (Train MSE)**
30 min	0.01898	0.01360	0.1213
60 min	0.03617	0.02481	0.1495
90 min	0.05058	0.03320	0.1763
120 min	0.06308	0.03959	0.1904
150 min	0.07427	0.04488	0.2112

events in the data.

6.3 N-BEATS

N-BEATS is the short form for *Neural basis expansion analysis for interpretable time series forecasting* [36]. The N-BEATS architecture consists of a basic block and double residual stack.

The key idea of N-BEATS is to generate a forecast and a backcast (that is, a reproduction of the input data) using fully connected blocks. One ore more blocks are connected in sequence, using residual connections, to form a stack; the final model is composed of a sequence of stacks, whose outputs are summed to produce the model output. This is shown in figure 6.3.

The input to a basic block is the history included in a lookback window of a particular length, X. There are two outputs of the basic block - block's estimate of the input called backcast, $\widehat{X}$ and block's forecast of a certain length, $\widehat{Y}$. Here, we have used the length of input is twice as the length of the forecast.

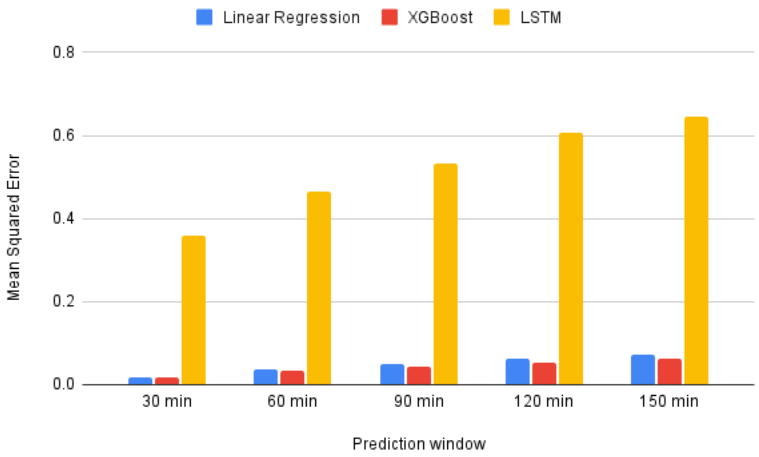


Figure 6.2: Comparison of the performance of LSTM with Linear Regression and XGBoost

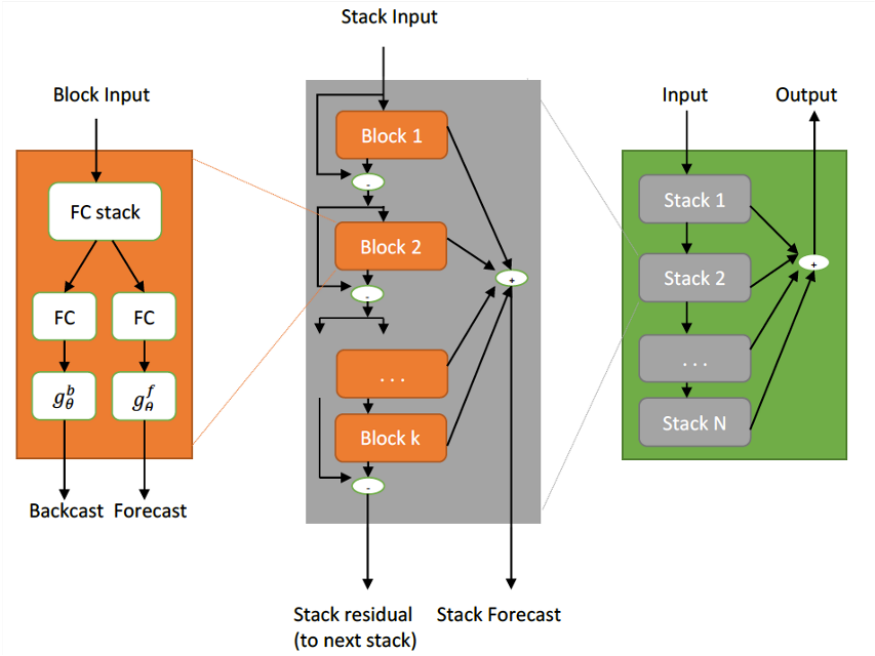


Figure 6.3: Architecture of N-BEATS

Table 6.4: Performance of LSTM vs Linear Regression vs XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**LSTM (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.35977	None
60 min	0.03589	0.03295	0.46356	None
90 min	0.05018	0.04479	0.53226	None
120 min	0.06216	0.05443	0.60702	None
150 min	0.07225	0.06186	0.64614	None

This basic block contains two parts. *i*) The waveform generator, $g_\theta : X^N \rightarrow Y^M$, maps N points in time domain to M points in forecast or backcast domain. *ii*) The forward predictor, ψ_ϕ^f, predicts the forward expansion parameters θ^f to optimize the accuracy of the partial forecast $\widehat{Y}$ and the backward predictor, ψ_ϕ^b, generates an estimate of X with the ultimate goal of helping the downstream blocks for forecasting by removing unnecessary components from the input. ψ_ϕ is a multi-layer fully connected (FC) neural network with ReLU non-linearity.

Each block learns and updates the residual from its previous block. Each block has two residual branches - one branch acts as backcast prediction branch of each layer and another one works as forecast prediction branch of each layer. Each block in the backcast prediction branch does sequential analysis of the input signal. That is how it makes the forecast job easier in the downstream blocks by removing unimportant portion of the input signal.

Finally hierarchical block stacking completes the architecture. The output of each layer in the forecast prediction branch is added to the stack level first and then to the overall network level.

Choice of g_θ makes the architecture either generic or interpretable. *a*) In the generic architecture, no time series specific knowledge is required. Here, g_θ is treated as a linear projection of the previous layer output. In this case, fully connected (FC) layers in the basic block learn the predictive decomposition of the partial forecast in the basis learned by the network. *b*) In the interpretable architecture, g_θ is incorporated with the trend (usually it is a monotonic function. In this architecture, a degree of small polynomial is designed to mimic the trend) and seasonality (In this architecture, a Fourier series is proposed to mimic the seasonal pattern, for example, regular, cyclical, recurring fluctuation) for the decomposition of the time series. That makes the stack outputs more easily interpretable.

6.3.1 Data Preparation

The shape of input for N-BEATS is a 2D tensor - (number of samples, lookback), where lookback is the length of the historical window for each sample. This input is fed into the architecture. The output is the maximum in the prediction window. We have used different lookback to analyze the behavior of the architecture.

6.3.2 Model Architecture

In this work, a small version of the N-BEATS architecture is used. The number of hidden layers in the basic building block is 2. In each layer the number of units are the same as the length of the prediction window, which is the number of timestep we are predicting in future. We have used 1 stack and in this stack there are 2 basic blocks.

6.3.3 Prediction

While training from data, the network learns all the parameters from the training data. Next these learned parameters are used to do prediction for multiple timestep in future. Then the maximum in the prediction window and the actual flux are used to calculate the mean squared error.

6.3.4 Experimental results

The numerical result shown in Table 6.6 presents the performance of N-BEATS in comparison with the best baselines, Linear Regression and XGBoost and the numerical result shown in Table 6.6 is graphically presented in Figure 6.4 for visualization purpose.

Table 6.5: Performance of N-BEATS vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**N-BEATS (Train MSE)**
30 min	0.01898	0.01360	0.02231
60 min	0.03617	0.02481	0.03802
90 min	0.05058	0.03320	0.05358
120 min	0.06308	0.03959	0.06777
150 min	0.07427	0.04488	0.08304

From the result we can see that the test MSE of N-BEATS is bigger than one of the simplest algorithms, Linear Regression. The poor performance of N-BEATS suggests that the fully connected layers may not capture the information from the data that is needed for the prediction task.

6.4 1D CNN Residual Learning

This technique consists of the following steps:

1. First we trained a linear regression model with the time series input and target flux.

Table 6.6: Performance of N-BEATS vs Linear Regression vs XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**N-BEATS (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.02221	None
60 min	0.03589	0.03295	0.03921	None
90 min	0.05018	0.04479	0.05553	None
120 min	0.06216	0.05443	0.06967	None
150 min	0.07225	0.06186	0.07844	None

2. Then we used the given time series as input to a linear regression model to predict the flux.

3. Next we calculated the difference between the prediction made by the linear regression model on the training time series input and the training target flux.

4. Finally this difference is used as target in the 1D CNN model to train the model.

Figure 6.5 presents the workflow for the 1D CNN Residual Learning model. For example, if X is the training time series input and Y is the training target flux, then X will the input for the linear regression model and let's assume, the prediction made by the linear regression model is $\overline{Y_l}$. Then the difference between the prediction and the actual flux is $|Y - \overline{Y_l}|$. This will be used as target to train the 1D CNN model.

6.4.1 Data Preparation

This approach involves two steps of data preparation. In the first step, the input shape for the linear model is a 2D tensor - (number of samples, lookback) and the output is a real

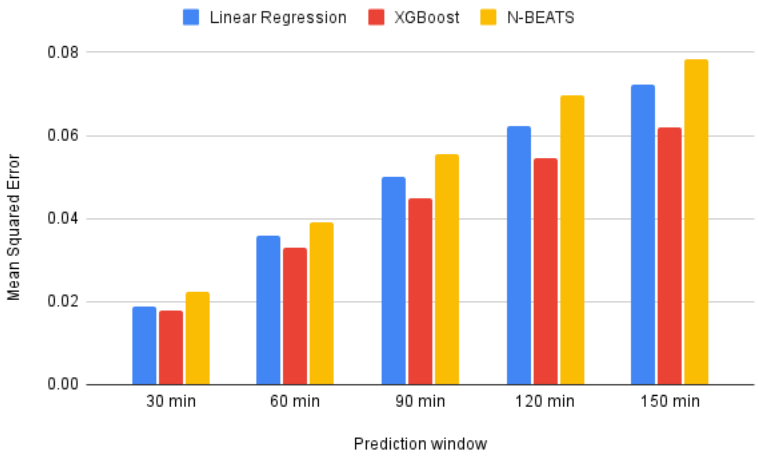


Figure 6.4: Comparison of the performance of N-BEATS with Linear Regression and XGBoost

Figure 6.5: Architecture of 1D CNN Residual Learning

number. For the next step, we have used 1D CNN model and the shape of input for 1D CNN is a 3D tensor - (number of samples, lookback, number of features) and the output is a real number.

6.4.2 Model Architecture and Prediction

We have used a linear regression model *section 4.1* and 1D CNN model *section 6.1*.

After training the linear model with the training data, prediction is made on the training data to calculate the difference between the actual training target and predicted training target which we have addressed as residual. This residual will be used as training target with the training features for the 1D CNN model to train the model. Then the 1D CNN model will predict the residual for the test features. The final result is the sum of this residual and the prediction from linear model on the test features. The performance metric is mean squared error.

6.4.3 Experimental results

The performance of residual learning is shown in Table 6.8 and is compared with the best non-machine learning baseline, Linear Regression and XGBoost. Figure 6.6 presents the graphical presentation of the numerical results shown in Table 6.8.
Figure 6.6 confirms that 1D CNN Residual Learning can outperform Linear Regression for a small prediction window by 11%. Better performance in 1D CNN Residual Learning suggests that residual learning might be significant for the solar flux prediction task. Figure 6.6 also shows that 1D CNN Residual Learning performs the same as Linear Regression when the prediction window is larger than 2 hours. In addition to that, it performs same as XGBoost for a small prediction window and the performance of this architecture gets worse than XGBoost when the prediction window becomes larger.

In Figure 6.7, we compare the performance of all the deep learning models with Linear Regression and XGBoost. Figure 6.7 shows that LSTM does very a poor job. So to analyze

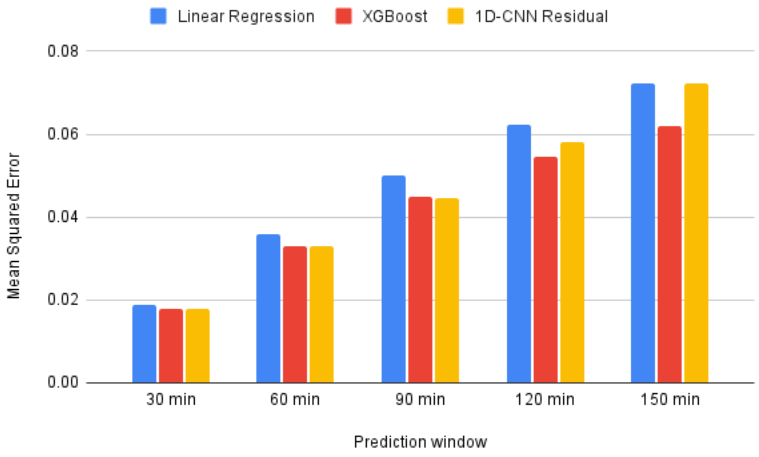


Figure 6.6: Comparison of the performance of 1D CNN Residual Learning with Linear Regression and XGBoost

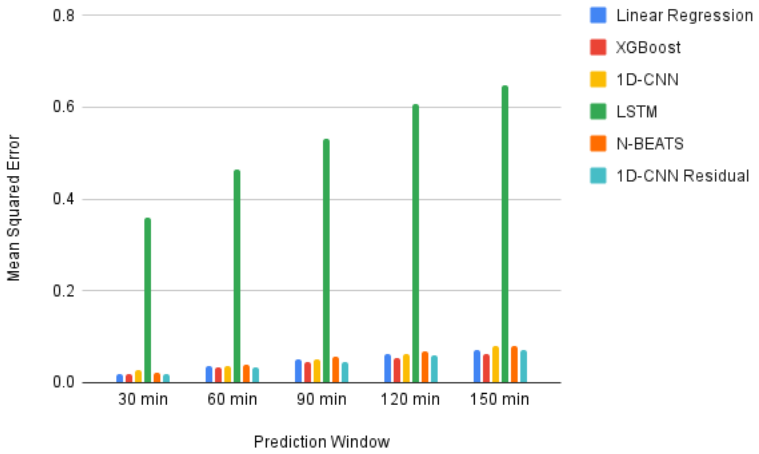


Figure 6.7: Comparison of the performance of all the neural networks with the baseline

Table 6.7: Performance of 1D CNN Residual Learning vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**1D CNN Residual Learning (Train MSE)**
30 min	0.01898	0.01360	0.01870
60 min	0.03617	0.02481	0.03440
90 min	0.05058	0.03320	0.04870
120 min	0.06308	0.03959	0.05910
150 min	0.07427	0.04488	0.07430

the result of the rest of the neural models in a better way we have presented the result in Figure 6.8 without the performance of LSTM.

The Figure 6.8 tells us that the performance of these neural models is not as good as Linear Regression for a larger prediction window. Neural networks are very popular because these models can learn hidden patterns in data and hence make a great job in the prediction task. But for this prediction task, the poor performance of these neural networks suggests that not having enough information about the solar higher class fluxes makes it difficult for these neural networks to learn important information about the data to do the prediction.

The execution time for all the deep models is shown in Figure 6.9. Figure 6.9 tells us that LSTM and XGBoost are computationally expensive and these models require more time as the prediction window gets bigger while the other models take almost linear time for different prediction window. To analyze the execution time of rest of the neural networks we

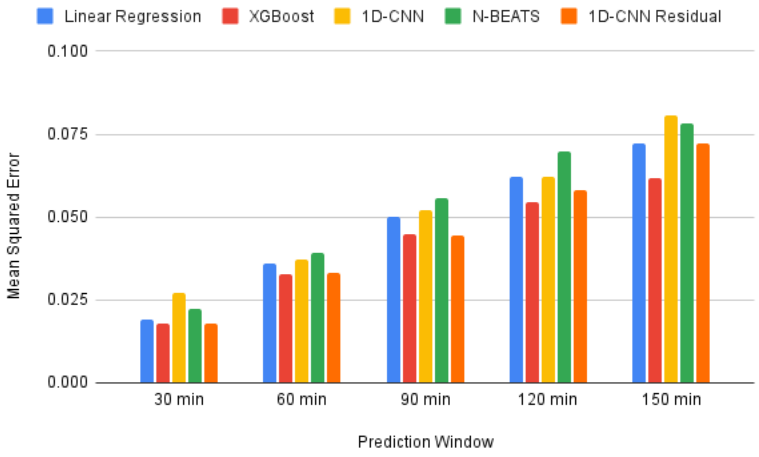


Figure 6.8: Performance of 1D CNN, N-BEATS and 1D CNN Residual Learning with the baseline, Linear Regression, XGBoost

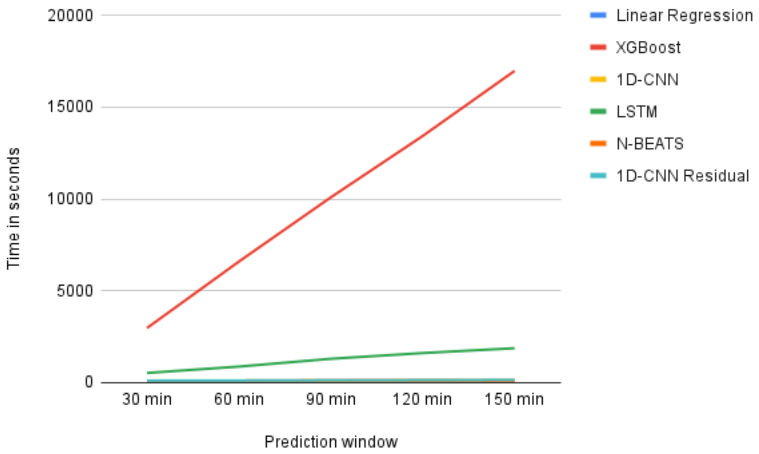


Figure 6.9: Comparison of the execution time of all the neural networks

Table 6.8: Performance of 1D CNN Residual Learning vs Linear Regression vs XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**1D CNN Residual Learning (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.01779	6.22%
60 min	0.03589	0.03295	0.03309	7.80%
90 min	0.05018	0.04479	0.04460	11.11%
120 min	0.06216	0.05443	0.05823	6.32%
150 min	0.07225	0.06186	0.07226	None

have presented the execution time of 1D CNN, N-BEATS and 1D CNN Residual Learning and Linear regression in Figure 6.10.

Figure 6.10 tells us that 1D CNN is the most time efficient neural model among all the neural models used here. Even though N-BEATS is a complex architecture, it takes almost the same time as 1D CNN because we used a smaller architecture of N-BEATS for our experiments. Linear Regression takes a very small amount of time compared to the neural models because it is very simple model compared to the neural networks. The execution time for training is longer for 1D Residual Learning compared to 1D CNN and N-BEATS.

6.5 Summary

- We experimented with small neural networks because there is no improvement in the performance with the complex neural networks.
- None of the neural networks could not outperform XGBoost.

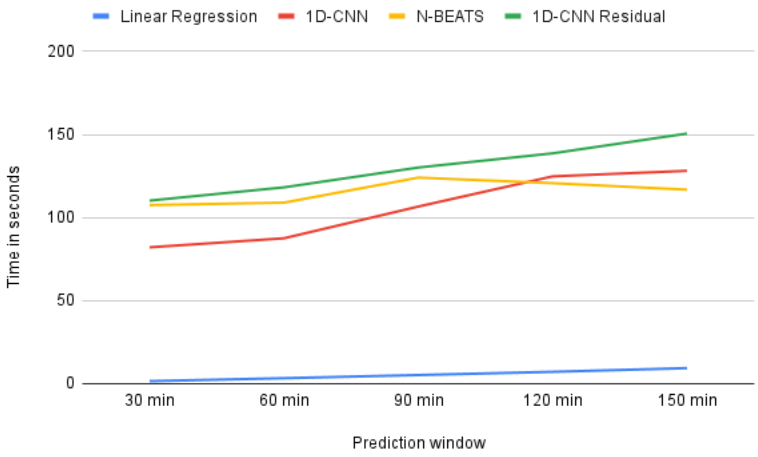


Figure 6.10: Comparison of the execution time of 1D CNN, N-BEATS and 1D CNN Residual Learning with Linear Regression

- The performance 1D CNN Residual Learning improves by 11% over Linear Regression when the prediction window is less than 2 hours.
- The performance 1D CNN Residual Learning is almost the same as XGBoost when the prediction window is less or equal to 2 hours.
- Better performance of 1D CNN Residual Learning and XGBoost suggests that the residual learning might be significant for the solar flux prediction.
- The performance of 1D CNN Residual Learning becomes the same as the performance of Linear Regression when the prediction window is larger than 2 hours.

Chapter 7

Shallow Models

In [11], the authors presented an architecture for time series classification that has shown very impressive results. On top of that, this architecture is very time efficient compared to the deep learning models. In this chapter, we are going to discuss this state-of-the-art technique. Also, we have developed some algorithms which are based on this state-of-the-art technique. We are going to discuss them as well.

7.1 ROCKET

ROCKET [11] stands for Random Convolution Kernel Transform. This is a time series classification or regression technique which is computationally efficient compared to deep learning techniques. To transform the time series, ROCKET generates random convolutional kernels with the following characteristics:

1. There is only one convolutional layer with a large number of kernels.
2. Kernel weights and biases are fixed after initialization; they are not trainable.
3. The parameters of each kernel are set randomly as follows:
   (a) Length is chosen from $7, 9, 11$ at random with equal probability.
   (b) Dilation is sampled on an exponential scale $\lfloor 2^y \rfloor$, where y is sampled from a uniform distribution, $d \sim U(0,\ R)$, where upper bound $R = log_2 \frac{l_i - 1}{l_k - 1}$, where l_i is the length of input and l_k is the length of the kernel.

(c) With kernels, padding will be used or not, that decision is made at random. In case of no padding, kernels match patterns in the central regions of time series and in case of padding, kernels match patterns at the beginning or end of the time series also.

(d) Weights are sampled from a normal distribution, $w \sim N(0,1)$.

(e) Bias is sampled from uniform distribution, $b \sim U(-1, 1)$. Only the positive values in the feature maps are used so that different biases can focus on different aspects of feature maps by shifting the values in the feature map in the positive or negative direction by a fixed amount.

4. Feature maps are computed by convolving the input with each of the kernels.

5. From each feature map, two features are derived: maximum value, which is global max pooling in conventional convolutional neural networks and proportion of positive values (PPV), which indicates the proportion of values in the feature map that are greater than zero.

6. Finally, a linear regressor is trained using these transformed features and the target values.

7.1.1 Model Architecture

The length of each kernel is chosen from $7, 9, 11$ at random. Dilation is sampled from an uniform distribution on exponential scale. Padding is chosen at random meaning for some kernels padding is used while for other kernels padding is not used. Weights and biased are sampled from a normal distribution and a uniform distribution respectively.

In the original implementation, 10000 kernels were used for the experiments. But after running several experiments with our dataset with different number of kernels, we have

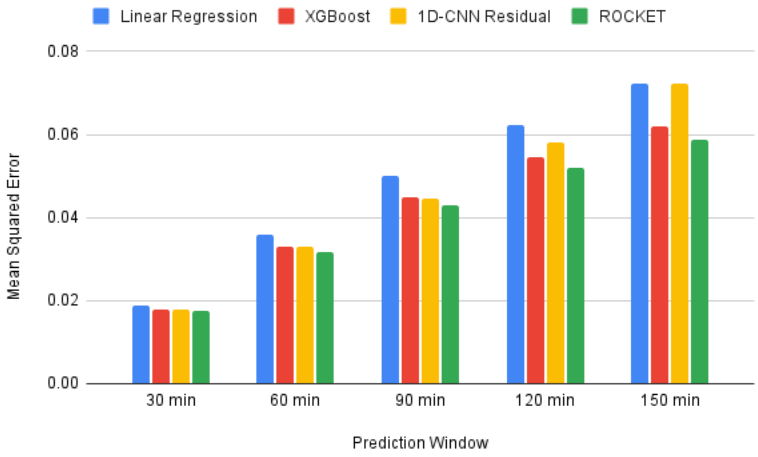


Figure 7.1: Comparison of ROCKET with the baseline, Linear Regression, XGBoost and 1D CNN Residual Learning

found that 200 kernels give us the best results and using more than 200 kernels is computationally expensive and the result does not improve.

7.1.2 Prediction

After training the model with the training data, the model predicts the maximum flux in the prediction window using the test data.

7.1.3 Experimental results

For the experiments, different lengths of lookback and prediction window are used. The following result shows the performance of the model in predicting the maximum flux in different prediction windows using the fluxes with different lengths of lookback. The metric is used to evaluate the performance is mean squared error.

We have compared the performance of ROCKET with the best linear model - Linear Regression and the best ensemble model - XGBoost. The numerical result of the comparison on the train set and on the test set are shown in the Table 7.1 and Table 7.2 respectively.

Table 7.1: Performance of ROCKET vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**ROCKET (Train MSE)**
30 min	0.01898	0.01360	0.01732
60 min	0.03617	0.02481	0.03163
90 min	0.05058	0.03320	0.04243
120 min	0.06308	0.03959	0.05128
150 min	0.07427	0.04488	0.05901

Figure 7.1 is the graphical representation of the Table 7.2 and it shows that ROCKET performs better than Linear Regression and XGBoost. ROCKET outperforms Linear Regression by 7% for a 30-minute prediction window. As the length of the prediction widow increases, the relative performance of ROCKET improves even more. For a 150-minutes prediction window, ROCKET improves by 19% over Linear Regression. ROCKET uses a variety of kernels that helps to capture patterns at different frequencies and scales. That eventually helps this model to perform better than Linear Regression and XGBoost.

7.2 MiniRocket

MiniRocket [12] is a modified version of ROCKET with more constraints in order to reduce the randomness from ROCKET by using fixed number of convolutional kernels with fixed length, dilation, padding, weights, bias.

1. MiniRocket uses kernels with a fixed length 9 (instead of random length chosen from $7, 9, 11$) and two values for the weights (instead of sampling weights from a

Table 7.2: Performance of ROCKET compared to Linear Regression and XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**ROCKET (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.01755	7.49%
60 min	0.03589	0.03295	0.03180	11.40%
90 min	0.05018	0.04479	0.04301	14.29%
120 min	0.06216	0.05443	0.05182	16.63%
120 min	0.07225	0.06186	0.05883	18.57%

normal distribution, *N(0,1)*). For example, if the two values of weights are p and q, then the full set of two-valued kernels of length 9 contains a subset of kernels with $1q$ (*e.g.*, $[p,p,p,p,p,p,p,p,q], [p,p,p,p,p,p,p,q,p]$), a subset of kernels with $2q$ (*e.g.*, $[p,p,p,p,p,p,p,q,q], [p,p,p,p,p,p,q,p,q]$), a subset of kernels with $3q$ (*e.g.*, $[p,p,p,p,p,p,q,q,q], [p,p,p,p,p,q,q,p,q]$) and so on. Now the total number of two-valued kernels is $2^9 = 512$. A subset of 84 of these 512 kernels are used in MiniRocket.

2. Two values of the weights are $p = -1$ and $q = 2$ and the constraint on these values is the sum of these weights should be zero meaning $q = -2p$. This constraints makes the kernels sensitive to the relative magnitude of the input only. This means the result of the convolutional operation between the input and the kernel is invariant to any shift to the input, *i.e.* $T \star K = (T \pm c) \star K$, where c is a constant.

3. As the length of the kernel is fixed, now the dilation only depends on the length of the input.

4. Half of the kernels use padding while other half use no padding and this is maintained

by using padding for the alternative kernels. In case of no padding, kernels match patterns in the central regions of time series and in case of padding, kernels match patterns at the beginning or end of the time series also, like ROCKET.

5. Bias values are sampled from the quantiles of the the result of convolutional operation between the kernel and a randomly selected training examples. For example, for a randomly selected training sample, T with a given kernel k and a dilation d, the convolution output is $k_d * T$ and the $[0.25, 0.5, 0.75]$ quantiles from the convolution output is used as bias that will be used in computing PPV.

Unlike ROCKET, MiniRocket extracts only one feature per kernel which is PPV. Finally, this transformed feature is used to train the regressor. As MiniRocket uses fixed number of kernels with fixed lenght, dilation, weights and bias, that makes the model even computationally less expensive than ROCKET.

7.2.1 Model Architecture

Here length of the kernel is 9. Dilation depends on the length of the input size. Half of the kernels use padding while the other half does not. $p = -1$ and $q = 2$ are used as weights. Biases are sampled from the quantiles of convolutional results.

We run experiments using different number of kernels. But to make the experiments comparable with ROCKET, we have presented the result using 200 kernels for MiniRocket.

The process to make the prediction is the same as of **ROCKET**.

7.2.2 Experimental results

We have compared the performance of the MiniRocket in forecasting in different prediction window with Linear Regression and XGBoost. Table 7.3 and Table 7.4 show the numerical results of the comparison on the train and test set respectively. For visualization purpose, we have presented the result shown in the Table 7.4 in the Figure 7.2.

Table 7.3: Performance of ROCKET vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**MiniRocket (Train MSE)**
30 min	0.01898	0.01360	0.02348
60 min	0.03617	0.02481	0.03903
90 min	0.05058	0.03320	0.04863
120 min	0.06308	0.03959	0.05877
150 min	0.07427	0.04488	0.06870

Figure 7.2 shows that MiniRocket outperforms Linear Regression when the prediction window is larger than 1 hour. Moreover, as the length of the prediction window increases, the percentage of improvement of MiniRocket also increases. For a 150-minutes prediction window, MiniRocket outperforms Linear Regression by 7%. But it could not outperform XGBoost. One reason might be MiniRocket only capture one feature PPV while processing the time series input and it does not consider maximum value unlike ROCKET. This result suggests that the maximum value may be an important feature while processing the time series input.

7.3 Modified ROCKET

Even though ROCKET is a time efficient algorithm compared to deep neural network, it does not fully utilize the resources of parallel computing. That is why we have implemented ROCKET from scratch using the Keras library to make it computationally more efficient. Modified ROCKET uses fixed kernels with different lengths, dilations and random weights

Table 7.4: Performance of MiniRocket compared to the baseline, Linear Regression and XGBoost in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**MiniRocket (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.02631	-3.87%
60 min	0.03589	0.03295	0.03955	-1.02%
90 min	0.05018	0.04479	0.05132	1.24%
120 min	0.06216	0.05443	0.05813	6.48%
120 min	0.07225	0.06186	0.06746	6.63%

and biases. Then the models extracts two features per kernel like ROCKET - proportion of positive values (PPV) and maximum value. Finally, a linear regressor is trained on the extracted features.

7.3.1 Model Architecture

Data is prepared exactly the same way as it is prepared in Linear regression *section 4.1.1.* Modified ROCKET uses kernels with lengths $3, 5, 7, 9$; dilation $1, 2, 3, 4, 6$; weights from a normal distribution with mean 0 and standard deviation 0.1 and biases from a uniform distribution with range $[-1, 1]$ and user defined number of filters in each convolutional layer. Then the combination of feature maps are used to extract two features - PPV and maximum value like ROCKET. This model uses 200 filters to extract 400 features like ROCKET. The process to make the prediction is the same as of ROCKET.

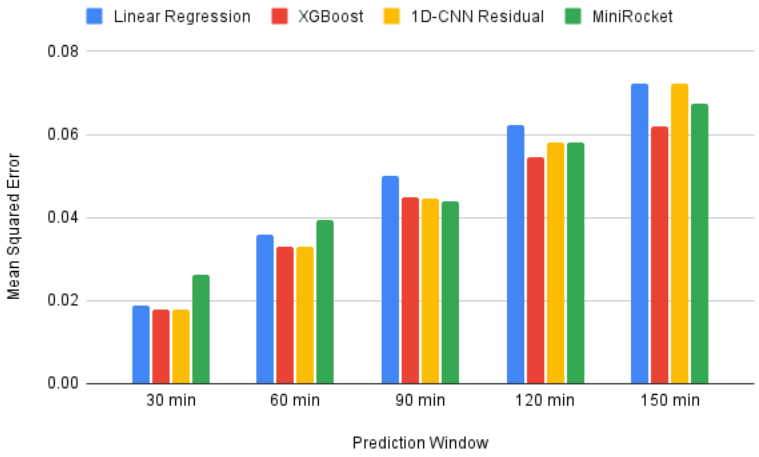


Figure 7.2: Graphical representation of the performance of MiniRocket

7.3.2 Experimental results

In this subsection, we present the performance of Modified ROCKET in forecasting flux and compare the results with Linear Regression and XGBoost. Table 7.7 and Table 7.8 shows the numerical results of these experiments on the train and the test set respectively. The graphical presentation of the Table 7.8 is shown in Figure 7.4.

Figure 7.4 tells us that Modified ROCKET outperforms Linear Regression. Like ROCKET and MiniRocket, the percentage of improvement of Modified ROCKET increases as the length of prediction window increases. Modified ROCKET outperforms Linear Regression by 7% for a 30-minutes prediction window and 14% for a 150-minutes prediction window. In addition to that it also shows us that it performs almost same as XGBoost. The result suggests that using fixed number of kernels with fixed length and dilation and random weights and bias helps the model to capture pattern in the data.

7.4 Ensemble ROCKET

This architecture is built on a collection of Modified ROCKET models. Each model has different number of kernels.

Table 7.5: Performance of Modified ROCKET vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**Modified Rocket (Train MSE)**
30 min	0.01898	0.01360	0.01752
60 min	0.03617	0.02481	0.03241
90 min	0.05058	0.03320	0.04443
120 min	0.06308	0.03959	0.05451
150 min	0.07427	0.04488	0.06297

7.4.1 Model Architecture

Data is prepared exactly the same way as it is prepared in Linear regression *section 4.1.1.* Each Modified ROCKET is built in the same way as described in *section 7.3.1.* The process to make the prediction is same as of ROCKET.

7.4.2 Experimental results

In this experiment, 10 modified ROCKET models with different number of kernels - 100, 160, 220, 280, 340, 400, 460, 520, 580 are used. The final result is the average of the results obtained by all these models. Table 7.7 and Table 7.8 present the numerical result of the ensemble ROCKET on the train and test set respectively. In these tables, the result of ensemble ROCKET is compared with the result of Linear Regression and XGBoost.

Figure 7.5 shows the performance of all kinds of ROCKET against Linear Regression and XGBoost. The results tell us that different Modified ROCKET models in Ensemble ROCKET learn to capture patterns with different frequencies and scales. That helps to

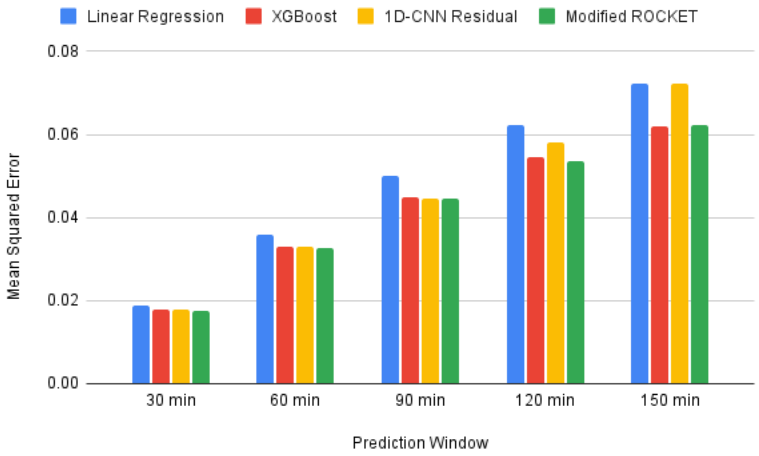


Figure 7.3: Graphical representation of the performance of Modified ROCKET

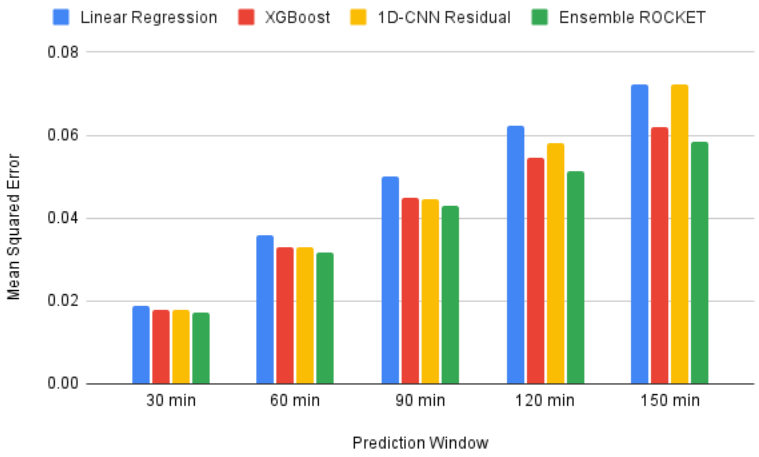


Figure 7.4: Graphical representation of the performance of Ensemble ROCKET

Table 7.6: Performance of Modified ROCKET in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**Modified ROCKET (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.01765	6.96%
60 min	0.03589	0.03295	0.03252	9.39%
90 min	0.05018	0.04479	0.04464	11.04%
120 min	0.06216	0.05443	0.05348	13.96%
120 min	0.07225	0.06186	0.06222	13.88%

improve the performance in the Ensemble ROCKET model.

From Figure 7.5, we can see that the Ensemble ROCKET outperforms all the models that were explored in this thesis. Like other versions of ROCKET, the Ensemble ROCKET performs even better for a bigger prediction window. Ensemble ROCKET outperforms Linear Regression by 9% for a 30-minute prediction window and 19% for a 150-minute prediction window.

We have also analyzed the execution time of these experiments. The result is shown in Figure 7.6. Figure 7.6 shows us that MiniRocket is the most time efficient algorithm. It takes almost same time as Linear Regression. Both ROCKET and Modified ROCKET needs more time as the length of prediction window increases. But the time for MiniRocket is pretty much same regardless the length of the prediction window.

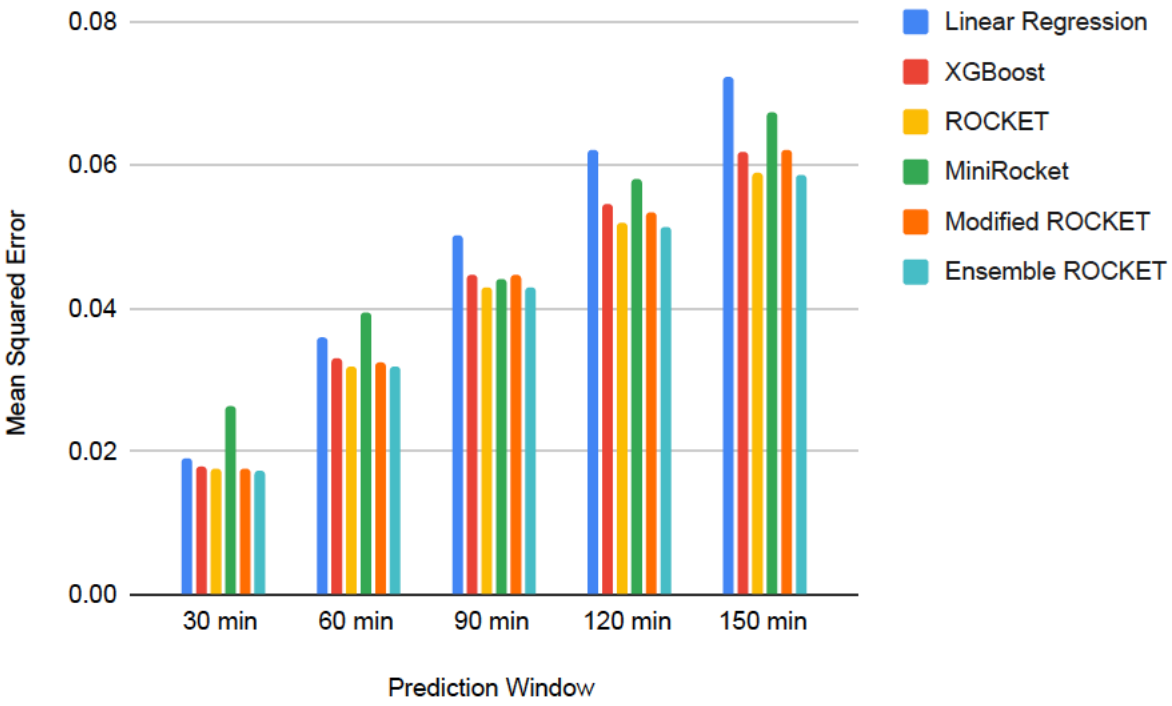


Figure 7.5: Graphical representation of the performance of all types of ROCKET with Linear Regression and XGBoost

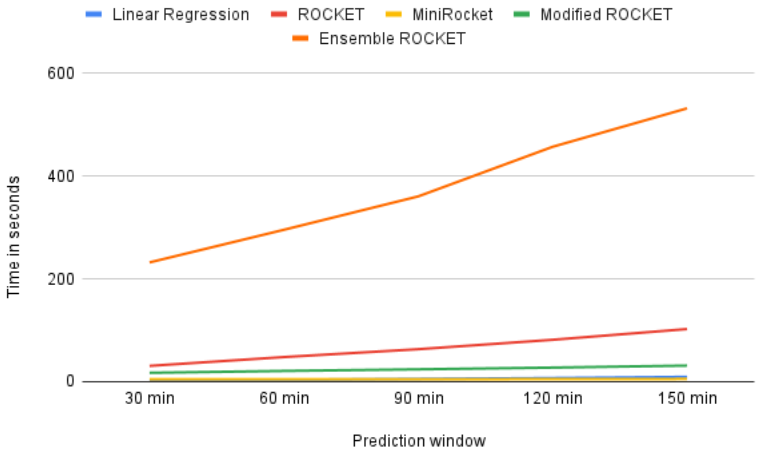


Figure 7.6: Graphical representation of the time required for all types of ROCKET with Linear Regression

Table 7.7: Performance of Ensemble ROCKET vs Linear Regression vs XGBoost on the training set

Prediction Window	**Linear Regression (Train MSE)**	**XGBoost Regression (Train MSE)**	**Ensemble Rocket (Train MSE)**
30 min	0.01898	0.01360	0.01751
60 min	0.03617	0.02481	0.03189
90 min	0.05058	0.03320	0.04308
120 min	0.06308	0.03959	0.05209
150 min	0.07427	0.04488	0.06003

7.5 Summary

- The original version of ROCKET outperforms both Linear Regression and XGBoost. This suggests that the idea of using a variety of kernels to capture patterns with different frequencies and scales is helpful for the solar X-ray flux prediction.

- As the prediction window increases, the percentage of improvement of ROCKET also keeps increasing. All versions of ROCKET behave the same in this case. The performance of ROCKET improves by 7% over Linear Regression for a 30-minute prediction window and 19% for a 150-minute prediction window.

- MiniRocket does not perform as well as ROCKET. The main difference between MiniRocket and ROCKET is that MiniRocket uses more restricted kernels than ROCKET and MiniRocket extracts one feature, PPV, per kernel. This suggests that randomness in kernels helps the kernels to capture the pattern and the maximum value (which is discarded in MiniRocket) might be an important feature for this prediction task.

Table 7.8: Performance of Ensemble ROCKET in forecasting fluxes

Prediction Window	**Linear Regression (Test MSE)**	**XGBoost Regression (Test MSE)**	**Ensemble ROCKET (Test MSE)**	**Improvement Percentage over Linear Regression**
30 min	0.01897	0.01790	0.01735	8.54%
60 min	0.03589	0.03295	0.03177	11.48%
90 min	0.05018	0.04479	0.04294	14.43%
120 min	0.06216	0.05443	0.05121	17.62%
120 min	0.07225	0.06186	0.5856	18.95%

- Modified ROCKET has less randomness in kernels than ROCKET and more randomness in kernel than MiniRocket. That is why Ensemble ROCKET meaning a collection of Modified ROCKET outperforms all versions of ROCKET.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Even though the dataset has more than 10 millions samples, the number of samples with high fluxes is very low. This lack of information makes the problem very challenging.

Most algorithms we explored in this thesis perform better than a simple baseline, but improvements were fairly modest.

A state-of-the-art deep learning model for time-series prediction could not outperform simple models, including linear regression.

The shallow convolutional model ROCKET outperforms all the models including the state-of-the-art deep model, N-BEATS. Moreover, ROCKET is not computationally expensive like the deep models.

XGBoost outperforms all models except for ROCKET, albeit at a significant computational cost. This suggests that residual learning is valuable in this domain.

8.2 Future Work

As ROCKET performs the best for this problem, this algorithm can be extended incorporating residual learning, additional features, and more sophisticated ensemble models.

As there is not enough information about the high fluxes, synthetic data can be generated using physics based models. Also, physics informed neural networks might be helpful for this problem.

The models developed and explored in this thesis, particularly the modified version of

ROCKET and ROCKET ensembles, can be evaluated with other datasets to determined to which degree they can be generalized.

References

[1] https://www.nasa.gov/content/goddard/the-difference-between-flares-and-cmes.

[2] A. K. Abed, R. Qahwaji, and A. Abed. The automated prediction of solar flares from sdo images using deep learning. *Advances in Space Research*, 67(8):2544–2557, 2021.

[3] A. Al-Ghraibah, L. E. Boucheron, and R. J. McAteer. A study of feature selection of magnetogram complexity features in an imbalanced solar flare prediction data-set. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 557–564. IEEE, 2015.

[4] H. Alfvén and P. Carlqvist. Currents in the solar atmosphere and a theory of solar flares. *Solar Physics*, 1(2):220–228, 1967.

[5] D. S. Bloomfield, P. A. Higgins, R. J. McAteer, and P. T. Gallagher. Toward reliable benchmarking of solar flare forecasting methods. *The Astrophysical Journal Letters*, 747(2):L41, 2012.

[6] M. G. Bobra and S. Couvidat. Solar flare prediction using sdo/hmi vector magnetic field data with a machine-learning algorithm. *The Astrophysical Journal*, 798(2):135, 2015.

[7] L. E. Boucheron, A. Al-Ghraibah, and R. J. McAteer. Prediction of solar flare size and time-to-flare using support vector machine regression. *The Astrophysical Journal*, 812(1):51, 2015.

[8] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[9] J. Cao and X. Lin. Study of hourly and daily solar irradiation forecast using diagonal recurrent wavelet neural networks. *Energy Conversion and Management*, 49(6):1396–1406, 2008.

[10] L. Comisso, M. Lingam, Y.-M. Huang, and A. Bhattacharjee. General theory of the plasmoid instability. *Physics of Plasmas*, 23(10):100702, 2016.

[11] A. Dempster, F. Petitjean, and G. I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[12] A. Dempster, D. F. Schmidt, and G. I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257, 2021.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[14] A. Dmitriev, Y. Minaeva, Y. Orlov, M. Riazantseva, and I. Veselovsky. Solar activity forecasting on 1999-2000 by means of artificial neural networks. *EGS XXIV General Assembly*, 22, 1999.

[15] J. Eastwood, E. Biffis, M. Hapgood, L. Green, M. Bisi, R. Bentley, R. Wicks, L.-A. McKinnell, M. Gibbs, and C. Burnett. The economic impact of space weather: Where do we stand? *Risk Analysis*, 37(2):206–218, 2017.

[16] D. Eck and J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103, 2002.

[17] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[18] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. D. Wierstra. A recurrent neural network for image generation. arxiv preprint. *arXiv preprint arXiv:1502.04623*, 2015.

[19] J. Heyvaerts, E. R. Priest, and D. M. Rust. An emerging flux model for the solar flare phenomenon. *The Astrophysical Journal*, 216:123–137, 1977.

[20] M. Janvier. Three-dimensional magnetic reconnection and its application to solar flares. *Journal of Plasma Physics*, 83(1), 2017.

[21] M. Janvier, G. Aulanier, and P. Démoulin. From coronal observations to mhd simulations, the building blocks for 3d models of solar flares (invited review). *Solar Physics*, 290(12):3425–3456, 2015.

[22] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[23] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] R. Li, Y. Cui, H. He, and H. Wang. Application of support vector machine combined with k-nearest neighbors in solar flare and solar proton events forecasting. *Advances in Space Research*, 42(9):1469–1474, 2008.

[26] R. Li, H.-N. Wang, H. He, Y.-M. Cui, and Z.-L. Du. Support vector machine combined with k-nearest neighbors for solar flare forecasting. *Chinese Journal of Astronomy and Astrophysics*, 7(3):441, 2007.

[27] X. Li, Y. Zheng, X. Wang, and L. Wang. Predicting solar flares using a novel deep convolutional neural network. *The Astrophysical Journal*, 891(1):10, 2020.

[28] S. Marra and F. C. Morabito. A new technique for solar activity forecasting using recurrent elman networks. *International Journal of Computational Intelligence*, 3(1):8–13, 2006.

[29] J. P. Mason and J. Hoeksema. Testing automated solar flare forecasting with 13 years of michelson doppler imager magnetograms. *The Astrophysical Journal*, 723(1):634, 2010.

[30] P. S. McIntosh. The classification of sunspot groups. *Solar Physics*, 125(2):251–267, 1990.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[32] T. A. Nagem, R. S. Qahwaji, S. S. Ipson, Z. Wang, and A. S. Al-Waisy. Deep learning technology for predicting solar flares from (geostationary operational environmental satellite) data. 2018.

[33] C. Napoli, F. Bonanno, and G. Capizzi. An hybrid neuro-wavelet approach for long-term prediction of solar wind. *Proceedings of the International Astronomical Union*, 6(S274):153–155, 2010.

[34] N. Nishizuka, K. Sugiura, Y. Kubo, M. Den, and M. Ishii. Deep flare net (defn) model for solar flare prediction. *The Astrophysical Journal*, 858(2):113, 2018.

[35] N. Nishizuka, K. Sugiura, Y. Kubo, M. Den, S. Watari, and M. Ishii. Solar flare prediction model with three machine-learning algorithms using ultraviolet brightening and vector magnetograms. *The Astrophysical Journal*, 835(2):156, 2017.

[36] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019.

[37] E. Park, Y.-J. Moon, S. Shin, K. Yi, D. Lim, H. Lee, and G. Shin. Application of the deep convolutional neural network to the forecast of solar flare occurrence using full-disk solar magnetograms. *The Astrophysical Journal*, 869(2):91, 2018.

[38] E. N. Parker. The solar-flare phenomenon and the theory of reconnection and annihiliation of magnetic fields. *The Astrophysical Journal Supplement Series*, 8:177, 1963.

[39] E. Priest. *Magnetohydrodynamics of the Sun*. Cambridge University Press, 2014.

[40] R. Qahwaji and T. Colak. Automatic short-term solar flare prediction using machine learning and sunspot associations. *Solar Physics*, 241(1):195–211, 2007.

[41] K. Shibata and T. Magara. Solar flares: magnetohydrodynamic processes. *Living Reviews in Solar Physics*, 8(1):6, 2011.

[42] H. Song, C. Tan, J. Jing, H. Wang, V. Yurchyshyn, and V. Abramenko. Statistical assessment of photospheric magnetic features in imminent solar flare predictions. *Solar Physics*, 254(1):101–125, 2009.

[43] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[44] X. Wang, Y. Chen, G. Toth, W. B. Manchester, T. I. Gombosi, A. O. Hero, Z. Jiao, H. Sun, M. Jin, and Y. Liu. Predicting solar flares with machine learning: Investigating solar cycle dependence. *The Astrophysical Journal*, 895(1):3, 2020.

[45] L. M. Winter and K. Balasubramaniam. Using the maximum x-ray flux ratio and x-ray background to predict solar flare class. *Space Weather*, 13(5):286–297, 2015.

[46] J.-G. Wu and H. Lundstedt. Geomagnetic storm predictions from solar wind data with the use of dynamic neural networks. *Journal of Geophysical Research: Space Physics*, 102(A7):14255–14268, 1997.

[47] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[48] A. Yona, T. Senjyu, T. FunabaShi, et al. Application of neural network to one-day—ahead 24 hours generating power forecasting for photovoltaic system [j]. *Intelligent Systems Applications to Power Systems*, 2007.

[49] J. Zeng and W. Qiao. Short-term solar power prediction using a support vector machine. *Renewable Energy*, 52:118–127, 2013.

Curriculum Vitae

Sumi Dey was born on February 20, 1988. She is one of the daughters of Nepal Chandra Dey and Mina Rani Dey. She graduated from The University of Dhaka, Bangladesh, in 2009 with a Bachelor of Science in Mathematics. In the Spring of 2014, she got herself admitted into the Graduate School of The University of Texas at El Paso. While pursuing a master's degree in Mathematics she worked as a Teaching Assistant at the Mathematical Sciences department. She graduated in Fall 2015 with a Master's of Science in Mathematics. In Spring 2016 she entered into the Computational Science Program to pursue her PhD.

Email address : sdey2@miners.utep.edu