Predictive modelling for geomagentic field

January 6, 2022

Oshita Saxena

Chennai Mathematical Institute

MSc Data Science

Predictive Analytics: Classification & Regression

Supervisor: Dr. Sushma Kumari

Contents

Abstract

The efficient transfer of energy from solar wind to Earth's magnetic field can cause massive geomagnetic storms. The resulting disturbances in the geomagnetic field can wreak havoc on GPS systems, satellite communication, and more. The Disturbance Storm-Time Index (Dst) is a measure of the severity of geomagnetic storms. Empirical models have been proposed around 1975 to forecast Dst solely from solar-wind observations at the Lagrangian (L1) position by satellites such as NASA's Advanced Composition Explorer (ACE). In this term paper, we will develop a neural network based model for forecasting Dst.

1 Introduction

Magnetic surveyors, government agencies, academic institutions, satellite operators, and power grid operators use the Dst index to analyze the strength and duration of geomagnetic storms. Several models were proposed for solar wind forecasting of Dst, including empirical, physics-based, and machine learning approaches. However, predicting extreme geomagnetic events remains particularly difficult and requires a robust solution that can handle raw real-time data streams under realistic conditions such as sensor malfunctions and noise. While the machine learning models generally perform better than models based on the other approaches, there is room to improve, especially when predicting extreme events and working on the raw, real-time data streams.

Problem Statement: The goal is to develop models for forecasting Dst that push the boundary of predictive performance, under operationally viable constraints, using specified real-time solar-wind data feeds.

2 Data set

The input data consists of solar wind measurements collected from two satellites: NASA's Advanced Composition Explorer (ACE) and NOAA's Deep Space Climate Observatory (DSCOVR). The Dst values are measured by four ground-based observatories near the equator. These values are then averaged to provide a measurement of Dst for any given hour. To ensure similar distributions between the training and test data, the data is

separated into three non-contiguous periods. All data are provided with a `period` and `timedelta` multi-index which indicates the relative timestep for each observation within a period, but not the real timestamp. The period identifiers and timedeltas are common across datasets. Three different time series datasets are provided as features, in addition to the Dst labels:

1. Solar wind data: This is the primary feature data collected from the ACE and DSCOVR satellites, recorded minutely.
2. Smoothed sunspot counts: The number of spots on the sun's disk, recorded monthly.
3. Coordinate satellite positions for ACE and DSCOVR, recorded daily.
4. Dst values averaged across the four stations, recorded hourly.

3 Feature generation

The data consists of three time series realizations, each of which are recorded at different frequencies. The primary features in the solar wind data measure the characteristics of interplanetary-magnetic-field (IMF) and plasma from solar winds. The smoothed sunspot count is the number of spots on the sun's disk. The Sun exhibits a well-known, periodic variation in the number of spots on its disk over a period of about 11 years, called a solar cycle. In general, large geomagnetic storms occur more frequently during the peak of these cycles. Sunspot numbers might allow for calibration of models to the solar cycle.

- `{bx,by,bz}_{gse,gsm}`: IMF {X,Y,Z}-component in {GSE,GSM} coordinate (nT-nanoTesla)
- `theta_{gse,gsm}`: IMF latitude in {GSE,GSM} coordinates (degrees)
- `phi_{gse,gsm}`: IMF longitude in {GSE,GSM} coordinates (degrees)
- `bt`: IMF component magnitude (nT)
- `density,speed,temperature`: Solar wind proton density (N/cm^3), bulk speed (km/s), ion temperature (Kelvin)
- `smoothed_ssn`: smoothed sunspot number

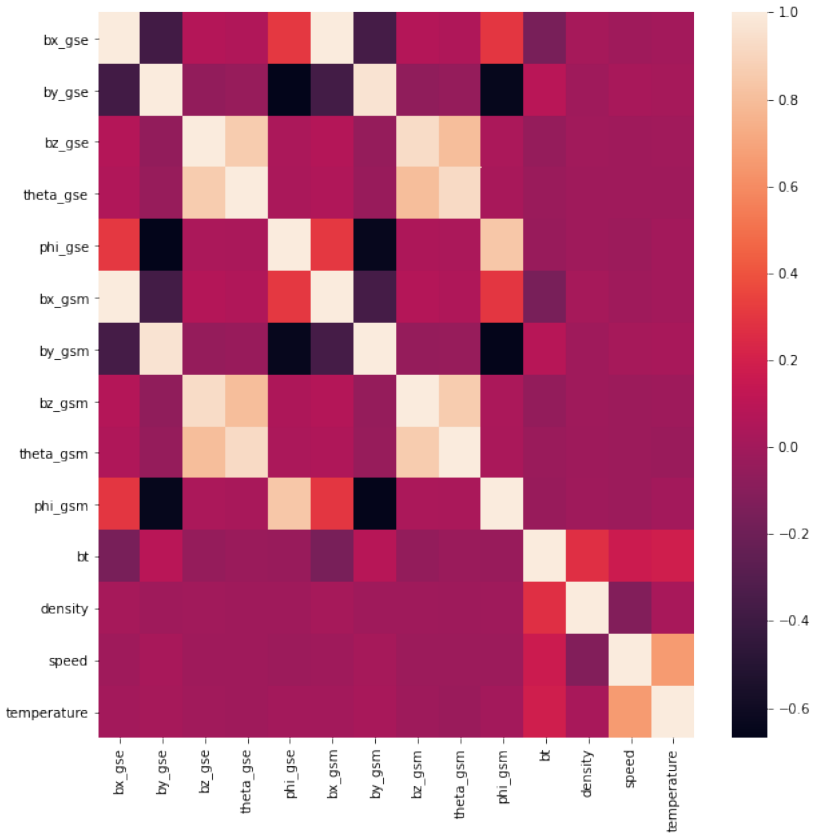

Figure 1: Correlation matrix of solar wind features

After observing the distribution of values across the three non-contiguous periods, `train_a`, `train_b`, `train_c`, it is seen that the mean and standard deviation of values in `train_a` is generally more intense than in the other two periods. But the boxplot of features for each period convey that the values belong to the same distribution.

We also notice that the values exist across very different scales. For instance, `temperature` values are quite high - reaching into the hundreds of thousands Kelvin. Meanwhile, IMF readings are fairly small values, and usually negative. The features are scaled by removing the mean and dividing by the variance.

Another observation is that many solar wind features have high correlation. This could lead to multicollinearity issues in the model. Hence, we select the following subset of features for model training based on the correlation matrix(Figure 1) such that these features have low correlation.

- `bt`

- temperature
- speed
- density
- phi_gsm
- {bx,by,bz}_gsm

Since, the features are provided at different frequencies and the dst labels are recorded hourly, the values are aggregated to the same frequency. The solar wind features are aggregated to the hour by taking the mean and standard deviation of each value for each hour. No aggregation is needed for smoothed_ssn as it is recorded monthly.

4 Methodology

For modelling the geomagnetic time series data, we will be using neural networks, LSTM in particular. LSTMs are a kind of recurrent neural network that especially suited to time series data due to their sequential structure. A better approach is to use Bidirectional Gated Recurring Unit. The Bi-GRU makes full use of both past and future information inside sequences and related features. To illustrate that the latter is a better approach, performance is evaluated for both LSTM and Bi-GRU.

4.1 LSTM

Long-Short-Term-Memory (LSTM) networks are capable of selectively remembering patterns for a long duration of time. The long-term memory is called the cell state. Intuitively, the LSTM cell consists of three gates, selective write, selective read and selective forget. The LSTM, like any sequential network, reads the data with increasing time steps.

For a time step t, the information is stored in a hidden state $s_t \in \mathbb{R}^d$. Assuming that we have computed the state s_{t-1} at timestep $t-1$ and now we want to provide new information x_t and compute the next hidden state s_t (Figure 2).

Selective Write

Instead of passing whole s_{t-1} to the next step, we usually want to pass only a fraction of it. This is done by giving each element in s_{t-1} weight between 0 and 1 indicating what

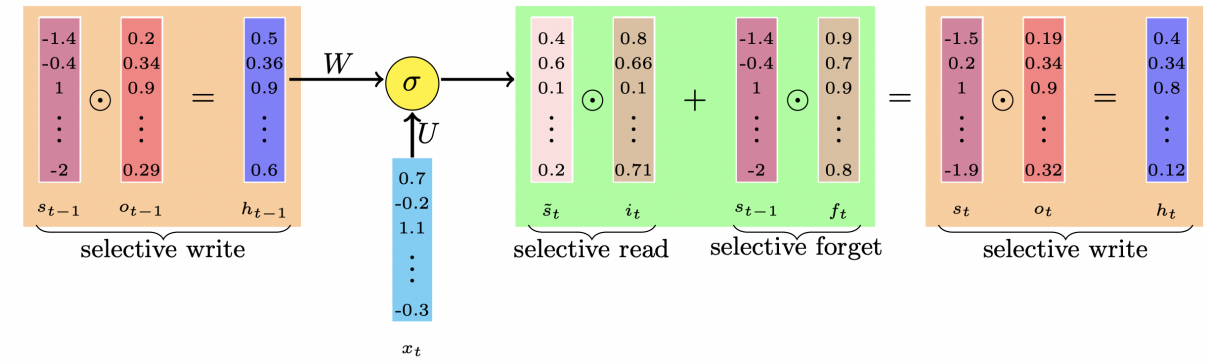

Figure 2: LSTM gates

part of the information is to be carried forward. These weights are stored in a vector o_{t-1}.

$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$

$$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$$

The logistic function σ ensures that the values are between 0 and 1. o_t is called the **Output Gate** as it decides how much to *write* to the next time step.

Selective Read

The feature h_{t-1} computed in the previous step is now used to compute the next time state. As the name suggests, here we attempt to selectively read the information from input x_t at time step t.

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

The value $i_t \odot \tilde{s}_t$ is the selective read information.

Selective Forget

Since we do not want the entire information in s_{t-1} to pass on to the next time step, a forget gate is introduced.

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

The RNN outputs $h_t = o_t \odot \sigma(s_t)$ for the calculation of the next state s_{t+1}.

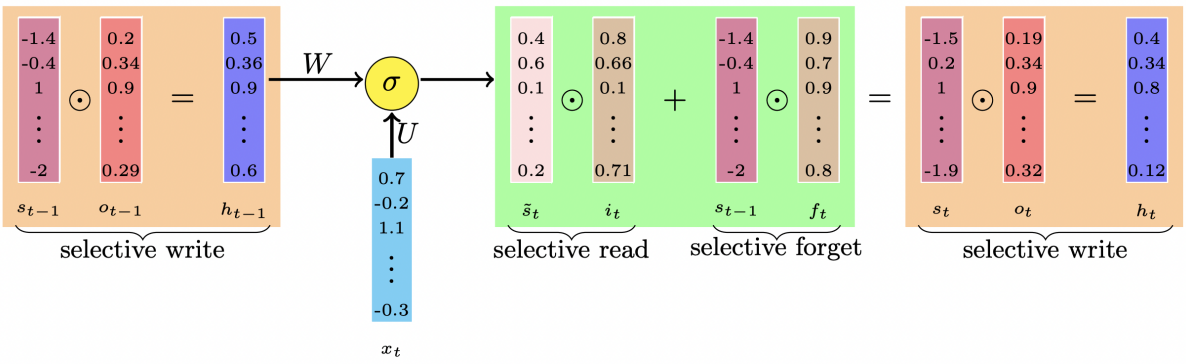

Figure 3: GRU

4.2 GRU

Gated Recurrent Unit is a variant of LSTM. GRU has no explicit forget gate. The forget gates and input gates are tied. The gates depend directly on s_{t-1} and not the intermediate h_{t-1} as in the case of LSTMs. The set of equations for GRUs are:

Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$$

4.3 Bi-LSTM and Bi-GRU

The networks that we have discussed capture information only from the past. But there are cases wherein we need to look far into the future to understand the current state. As the name suggests, bidirectional RNNs combine an RNN that moves forward through time, beginning from the start of the sequence, with another RNN that moves backward through time, beginning from the end of the sequence. The structure of a two layered bi-GRU is as shown in Figure 4. bi-LSTM has a similar structure with GRUs replaced by LSTMs.

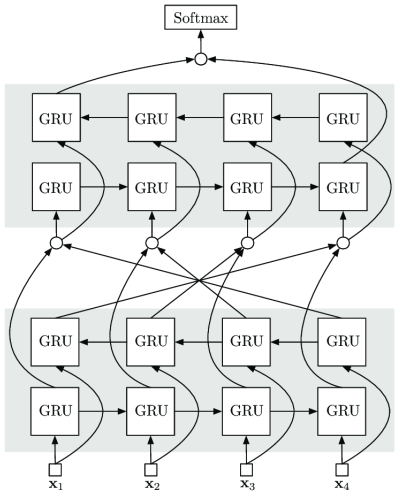

Figure 4: Bi-GRU

5 Model

Before building our model we have to separate our data into sequences and batches.

- `timesteps`: This determines the sequence length or how many timesteps in the past to use to predict each step at t0 and t1. The data is aggregated hourly, so timesteps is equal to the number of hours we want to use for each prediction.
- `batch_size`: This determines the number of samples to work through before a model's parameters are updated.

Initially we choose 32 timesteps per sequence and 32 sequences per batch. We use these numbers to separate our training data and labels into batches of sequences that will be fed into the model. To make sure that the sequences don't span across periods, we iterate through our periods and generate a time series dataset for each one.

LSTM

The design of our LSTM network is a simple sequential model with one hidden LSTM layer and one output layer with 2 output values (t0 and t1). We call The hyperparameters used are:

- `n_epochs`: This determines the number of complete passes the model takes through the training data. We choose 20 epochs.

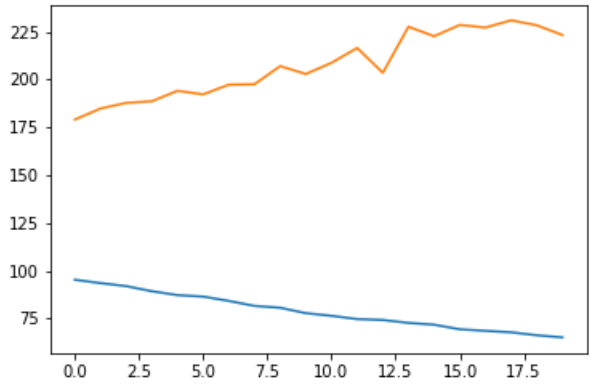

Figure 5: Convergence of `model1`

- `n_neurons`: The number of hidden neurons in each layer. We use 512.
- `dropout`: This regularizes by randomly “ignoring" a dropout fraction of a layer’s neurons during each pass through the network during training, so that no particular neuron overfits its input. We use 0.4.

The convergence of the model is as shown in Figure 5, which is plot of training loss vs validation loss. The model is just starting to converge. Next we introduce bi-RNNs in our model.

Bi-RNN

Next we move on to a model containing bi-GRU and bi-LSTM. We call this `model2`. The model structure consists of a bi-LSTM connected to a bi-GRU followed by three dense layers connected to the GRU through a flatten layer. And a dense output layer. The hyperparameters used are as follows:

- Number of neurons in LSTM layer: 192
- Number of neurons in GRU layer: 576
- Number of neurons in dense layers: 96, 128, 64 respectively in the the three layers
- Number of neurons in dense output layer: 2

The convergence of the model is as shown in Figure 6. It is clear that the model with bidirectional RNNs gives better results.

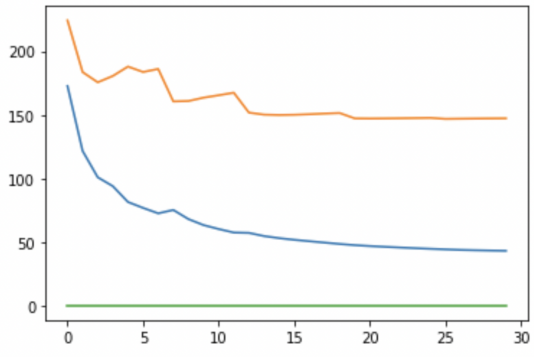

Figure 6: Convergence of `model2`

6 Performance metric

Performance is evaluated according to Root Mean Squared Error (RMSE). RMSE will be calculated on t_0 and $t+1$ simultaneously.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}$$

where:

$\hat{y}_i$ is the estimated Dst value for t_0 and $t+1$

y_i is the recorded Dst for t_0 and $t+1$

N is the number of samples

7 Results

The test RMSE for the first model with simple LSTM is 14.24. And the second gave an RMSE equal to 12.22. Clearly, `model2` gives better results.

8 References

1. https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

2. http://colah.github.io/posts/2015-08-Understanding-LSTMs/