**Final Project: Solar Wind Delay Prediction**

**CS 588 – Big Data Computing**

Prudvi Gadireddy

November 29, 2021

**Table of Contents**

[**Part 1: Introduction** 3](#_Toc89119012)

[**Overview of Data, Objective Evaluation, & Hypothesis** 3](#_Toc89119013)

[**5V Model** 3](#_Toc89119014)

[**Part 2: Implementation** 5](#_Toc89119015)

[**Preprocessing Steps** 5](#_Toc89119016)

[**Model Selection and Feature Analysis** 5](#_Toc89119017)

[**Part 3: Performance Evaluation** 6](#_Toc89119018)

[**Accuracy and RSME** 6](#_Toc89119019)

[**Part 4: Conclusions & References** 7](#_Toc89119020)

[**Analysis on Results** 7](#_Toc89119021)

[**References** 7](#_Toc89119022)

[**Part 5: Code Appendix** 8](#_Toc89119023)

[**Code** 8](#_Toc89119024)

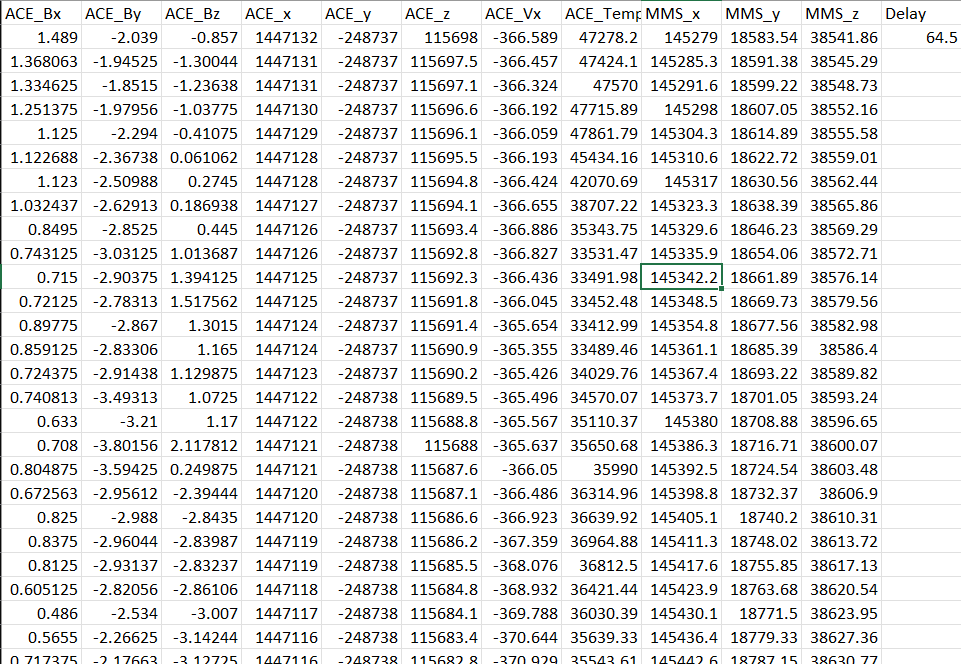
# 

# **Part 1: Introduction**

## **Overview of Data, Objective Evaluation, & Hypothesis**

Solar wind refers to the continuous flow of particles, mainly protons/electrons as plasma, emanating from the Sun[1]. The behavior of this flow, and its embedded magnetic field, has a direct impact on space weather, which affects Earth as well. For example, high intensity wind can create geomagnetic storms which may affect satellites and other electronics on Earth, with possibly dangerous results[2]. Therefore, predicting space weather, as it relates to the solar wind, is vital. One aspect of this objective is determining how long some set of solar wind will take to propagate a given distance, such as from some the L1 point to the Earth. Since this solar wind propagation delay is not easily calculable, machine learning techniques may be able to better predict this delay value given aspects of the wind as the input.

The training and testing data used for the model is provided by Dr. Samira Tasnim and the Center for Space Plasma and Aeronomic Research (CSPAR) at UAH; the data was originally recorded by a satellite. The CSV file had 56538 lines of data organized into 698 separate sets of solar wind data; each set has about 80 records (rows) of a solar wind instance’s attributes measured over time. For example, lines 1 to 81 is one block. The next block is lines 82 to 162. Eleven attributes are recorded, which include the speed and magnetic field intensity in 3 dimensions, as well as temperature. The 12th attribute, "delay" is what is being calculated. Each block resolves to a single value, called "delay". For example, the first block resolves to 64.5. Using linear regression on the data, predictive analysis will be attempted to accurately predict the delay value for each solar wind instance. An example of the solar wind instance (block) is shown below:



## **5V Model**

To accurately predict delay, a lot of data, composed of many attributes, needs to be collected. The data file used for this project has about 700 different blocks of data, each having 80 records with 11 measured attributes. Thus, the data file is somewhat large, having about 57000 records in total; this aspect fulfills the Volume and Variety requirements of big data. Moreover, in theoretical real-time applications concerning geomagnetic storm preparation, this data is sent by a satellite rapidly in packets, which requires quick analysis to determine the delay value so that action can be taken. Solar wind, as with weather on Earth, is highly chaotic and volatile, changing rapidly. These aspects satisfy the Velocity and Variability requirement of big data. Given the potential importance of this data, the Veracity of this data is key as well since false data can lead to false predictions and hazardous results. In a similar vein, the Value of this data is evident as well, not only for safety, but for researching the behavior of the solar wind.

# **Part 2: Implementation**

## **Preprocessing Steps**

The initial data used for this project was already preprocessed and organized, so minimal steps were taken before implementation. The data was very clean to start with, having no duplicates or nonsense data. However, a few blocks had only 78 records each, which would make processing them difficult; since 700 other blocks were still valid, these 6 dissimilar blocks were eliminated. Furthermore, the initial given data was spread out over about 20 different CSV files; these were compounded into one, and extraneous column headers were removed.

## **Model Selection and Feature Analysis**

Regression, a supervised learning approach, was chosen for this project. Since this data is numerical and the output is not easily classifiable into categories, linear regression seemed to be the best fit, since classification/clustering were not applicable. However, since the objective is to resolve a 2-D block into a single value, instead of a single record to a value, a modified approach was used. After splitting the 57000 records into train/test sets, training commenced by processing each record independently, with the given output value being the delay value of the records parent block. During prediction of a whole block’s delay value, each constituent record’s delay was calculated and then averaged to get the final delay. Furthermore, 3 different train/test split ratios were used to experiment and see which would result in the optimal outcome and least time. The ratios used were 20%, 50%, and 70%.

Given the size of this dataset, processing time will be extensive if some feature reduction did not take place. A correlation heatmap was initially constructed to analyze each variable’s relationships; the heatmap is shown below. No variable exhibited a high degree of correlation with another, besides itself, so all variables were kept and processed.

Chart

Description automatically generated

# **Part 3: Performance Evaluation**

## **Accuracy and RSME**

Two metrics were used to quantify the effectiveness of the linear regression approach. Accuracy, or the model score, measured, on average, how close predictions were percentage wise to the actual value. Root mean squared error (RSME) is an extension of this and measures the general quality of the regression predictor by measuring the variance of the predicted values. A high model score and low RSME is ideal. Since 3 different train/test split ratios were used, the accuracy and RMSE of each split was calculated and compiled into four bar charts. First, the accuracy and RMSE of the training sets is shown below; as can be expected, the 70% training split had the best results overall.

Chart, bar chart

Description automatically generated

Next the accuracy and RMSE of the test sets are shown below; unexpectedly, the accuracy/model score went down as training/test ratio went up. The RMSE behaved as expected.

Chart, bar chart

Description automatically generated

# **Part 4: Conclusions & References**

## **Analysis on Results**

While the linear regression approach is easy to compute or implement, it is not particularly suited for or performed well in predicting the delay. The lowest achieved RSME value was about 4, which is moderately unsuccessful, due the importance of predicting the delay as accurately as possible in order to deal with and prepare for solar wind impacts. While classification may not be a straightforward approach, it may be possible to define categories as different delay ranges and implement it that way. The best approach may be to implement a neural net that can resolve the solar wind instance block to a delay value in a more logical manner.

## **References**

* 1. <https://www.swpc.noaa.gov/phenomena/solar-wind>
  2. https://arxiv.org/abs/2106.14513

# 

# **Part 5: Code Appendix**

## **Code**

import pandas as pd

import seaborn as sns

import numpy as np

sns.set(rc={'figure.figsize':(11.7,8.27)})

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

import matplotlib.pyplot as plt

df=pd.read\_csv("Final Data Sheet.csv")

df.head()

# These steps are required to remove the column names after every block (as they are not required)

# Start with the first column that is replaced by zero to get the exact index position for every row with column name

df["ACE\_Bx"]=df["ACE\_Bx"].replace("ACE\_Bx",0)

# and in the code below every other row with column name is replaced by zero (to identify )

df\_emp=df[df["ACE\_Bx"] == 0].replace(df.columns[1:],0)

df\_emp

# Remove the rows with zero ( the column names)

df=df.drop(index=df\_emp.index.tolist()).reset\_index(drop=True)

# Convert the data types to float

print(df.dtypes)

for x in df.columns.tolist():

df[f"{x}"]=df[f"{x}"].astype(float)

print(df.dtypes)

df.insert(11, "Block", "Block1")

# add the block column just for identifcation

# the block can also be used to avergae if required

# LOGIC TO FILL IN THE DELAY\_VALUE FOR EVERY BLOCK

initial\_index=0

block\_index=1

final\_index=80

while final\_index <= df.shape[0]:

delay\_value=df.iloc[initial\_index,-1]

df.iloc[initial\_index:final\_index,-1]=delay\_value

df.iloc[initial\_index:final\_index,-2]=f"Block {block\_index}"

initial\_index+=80

final\_index+=80

block\_index+=1

df.tail()

# Now the values are filled and the preprocessing is done

# We can proceed with adding the heatmap and traning the models

df.isna().sum()

# pearson correlation with the pandas corr()

features = df.iloc[:,0:-2]

sns.heatmap(features.corr(method="pearson"),annot=True,linewidth=0.5)

# function to compute Average Delay

def computeAverageDelay(array):

avg=[]

initial\_index=0

final\_index=80

while final\_index <= array.shape[0]:

avg.append(array[initial\_index:final\_index].mean())

initial\_index+=80

final\_index+=80

return (np.array(avg))

# evaluate

def evaluate(X\_train,y\_train,y\_train\_avg,y\_pred\_avg,model):

accuracy=abs(model.score(X\_train,y\_train))

rmse = abs(metrics.mean\_squared\_error(y\_train\_avg,y\_pred\_avg,squared=False))

return (accuracy,rmse)

# get and X and y

X=df.iloc[:,:-2]

y=df.iloc[:,-1]

#Shuffle needs to be false here to make sure that it is divided block wise

# DIVIDE DATA INTO TRAIN AND TEST SPLIT

# OF 20 TRAIN AND 80 TEST

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, train\_size=0.20, shuffle=False)

LR20=LinearRegression()

LR20.fit(X\_train,y\_train)

y\_pred\_train = LR20.predict(X\_train)

y\_pred\_train\_avg= computeAverageDelay(y\_pred\_train)

y\_pred\_test = LR20.predict(X\_test)

y\_pred\_test\_avg= computeAverageDelay(y\_pred\_test)

y\_train\_avg =computeAverageDelay(y\_train)

y\_test\_avg =computeAverageDelay(y\_test)

lr20\_accuracy\_train, lr20\_rmse\_train= evaluate(X\_train,y\_train,

y\_train\_avg,y\_pred\_train\_avg,LR20)

lr20\_accuracy\_test, lr20\_rmse\_test= evaluate(X\_test,y\_test,

y\_test\_avg,y\_pred\_test\_avg,LR20)

# Next do the same process for 50 percent train size

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, train\_size=0.50, shuffle=False)

LR50=LinearRegression()

LR50.fit(X\_train,y\_train)

y\_pred\_train = LR50.predict(X\_train)

y\_pred\_train\_avg= computeAverageDelay(y\_pred\_train)

y\_pred\_test = LR50.predict(X\_test)

y\_pred\_test\_avg= computeAverageDelay(y\_pred\_test)

y\_train\_avg =computeAverageDelay(y\_train)

y\_test\_avg =computeAverageDelay(y\_test)

lr50\_accuracy\_train, lr50\_rmse\_train= evaluate(X\_train,y\_train,

y\_train\_avg,y\_pred\_train\_avg,LR50)

lr50\_accuracy\_test, lr50\_rmse\_test= evaluate(X\_test,y\_test,

y\_test\_avg,y\_pred\_test\_avg,LR50)

# same process for the 70 percent data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, train\_size=0.70, shuffle=False)

LR70=LinearRegression()

LR70.fit(X\_train,y\_train)

y\_pred\_train = LR70.predict(X\_train)

y\_pred\_train\_avg= computeAverageDelay(y\_pred\_train)

y\_pred\_test = LR70.predict(X\_test)

y\_pred\_test\_avg= computeAverageDelay(y\_pred\_test)

y\_train\_avg =computeAverageDelay(y\_train)

y\_test\_avg =computeAverageDelay(y\_test)

lr70\_accuracy\_train, lr70\_rmse\_train= evaluate(X\_train,y\_train,

y\_train\_avg,y\_pred\_train\_avg,LR70)

lr70\_accuracy\_test, lr70\_rmse\_test= evaluate(X\_test,y\_test,

y\_test\_avg,y\_pred\_test\_avg,LR70)

# plot the accuracy and rmse for the models

# first create the dataframe for it

trainSet\_metrics = pd.DataFrame({"Train\_Size":["20","50","70"],

"Accuracy":[lr20\_accuracy\_train,lr50\_accuracy\_train,lr70\_accuracy\_train],

"RMSE":[lr20\_rmse\_train,lr50\_rmse\_train,lr70\_rmse\_train]})

# use log scale to plot on graph

testSet\_metrics = pd.DataFrame({"Train\_Size":["20","50","70"],

"Accuracy":np.log([lr20\_accuracy\_test,lr50\_accuracy\_test,lr70\_accuracy\_test]),

"RMSE":np.log([lr20\_rmse\_test,lr50\_rmse\_test,lr70\_rmse\_test])})

fig, (ax1, ax2) = plt.subplots(1,2,figsize=(11.7,8.27))

sns.barplot(x="Train\_Size",y="Accuracy",data=trainSet\_metrics,ax=ax1).set\_title("Accuracy on train set")

sns.barplot(x="Train\_Size",y="RMSE",data=trainSet\_metrics,ax=ax2).set\_title("RMSE on train set")

fig, (ax1, ax2) = plt.subplots(1,2,figsize=(11.7,8.27))

sns.barplot(x="Train\_Size",y="Accuracy",data=testSet\_metrics,ax=ax1).set\_title("Accuracy on test set")

sns.barplot(x="Train\_Size",y="RMSE",data=testSet\_metrics,ax=ax2).set\_title("RMSE on test set")