Machine Learning

Coursework 2

Mars Weather Data Set

Süleyman Kalkan
University of Bradford
UBN: 21042430

Date: 20/05/2022

Introduction

The curiosity of the human mind and the possibility of settlement on other planets to decrease the likelihood of human extinction have acted as a catalyst in the colonization mission of the planet Mars. Exploration, colonization and human missions to the planet are being supported by many public space agencies. Although there are several factors like toxic soil, low gravity, radiation exposures etc. that rule out the possibility of colonization, the presence of polar ice caps gives abundant hope to scientists towards making Mars habitable. Colonizing the planet also considers factors like atmosphere, soil, water content etc., and there seems to be an ongoing debate on how to make the planet habitable for mankind. To strengthen or weaken the claim, there is a necessity to explore many other factors that may contribute to Mars' colonization in the future. Weather is one such factor worth exploring. (Researchgate,2021)

In this report, I will present training on machine learning algorithms on the Mars Weather Dataset.

Background

Machine learning systems are also commonly used in space applications to approximate complex representations of the real world. For instance, when analysing massive amounts of Earth observation data or telemetry data from spacecraft, Machine learning plays an important role. Potential applications of AI are also being thoroughly investigated in satellite operations, in particular, to support the operation of large satellite constellations, including relative positioning, communication and end-of-life management.

In addition, it is becoming more common to find ML systems analysing the massive amount of data that comes from each space mission. The data from some Mars rovers are being transmitted using AI, and these rovers have even been taught how to navigate by themselves. (The European Space Agency, 2020)

Methodology and Data

In this report, we will be working on Mars Weather Dataset and try to produce machine learning for the recognition for the prediction of mars weather temperature. When implementing Machine learning techniques to a dataset to achieve a goal, One of the most crucial things to do is to investigate the nature of the data itself.

The dataset was measured and transmitted via the REMS on board the Curiosity Rover. The data was made publicly available by NASA. The data is representing Mars Weather conditions such as temperature, pressure, etc from 7/08/2012 to 27/02/2018.

Dataset Attribute Descriptions:

1. **id**: The identification number of a single transmission

2. **terrestrial_date**: The date on Earth (formatted as mm/dd/yy).

3. **ls**: The solar longitude or the Mars-Sun angle, measured from the Northern Hemisphere. In the Northern Hemisphere, the spring equinox is when ls = 0. Since Curiosity is in the Southern Hemisphere, the following ls values are of importance:
   • ls = 0: autumnal equinox
   • ls = 90 : winter solstice
   • ls = 180 : spring equinox
   • ls = 270 : summer solstice

4. **month**: The Martian Month. Similarly to Earth, Martian time can be divided into 12 months.

5. **min_temp**: The minimum temperature (in °C) observed during a single Martian sol.

6. **max_temp**: The maximum temperature (in °C) observed during a single Martian sol.

7. **pressure**: The atmospheric pressure (Pa) in Curiosity's location on Mars.

8. **wind_speed**: The average wind speed (m/s) measured in a single sol. Note: Wind Speed data has not been transmitted to Earth since Sol 1485. Missing values are coded as NaN.

9. **atmo_opacity**: Description of the overall weather conditions on Mars for a given sol based on atmospheric opacity (e.g., Sunny).

Let's import the dataset from CSV and have a look at its data types.

```
import pandas as pandas

data = pandas.read_csv('mars-weather.csv')

data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1894 entries, 0 to 1893
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                1894 non-null   int64
 1   terrestrial_date  1894 non-null   object
 2   sol               1894 non-null   int64
 3   ls                1894 non-null   int64
 4   month             1894 non-null   object
 5   min_temp          1867 non-null   float64
 6   max_temp          1867 non-null   float64
 7   pressure          1867 non-null   float64
 8   wind_speed        0 non-null      float64
 9   atmo_opacity      1894 non-null   object
dtypes: float64(4), int64(3), object(3)
memory usage: 148.1+ KB

Process finished with exit code 0
```

Some of the columns are non-numeric, let's investigate these columns.

```
import pandas as pandas

data = pandas.read_csv('mars-weather.csv')

print(data.select_dtypes(include='object').sample(10))
```

```
     terrestrial_date     month atmo_opacity
213        2017-07-22   Month 2        Sunny
1749       2013-01-09   Month 9        Sunny
554        2016-08-05   Month 7        Sunny
1458       2013-12-26   Month 3        Sunny
197        2017-08-07   Month 2        Sunny
130        2017-10-16   Month 3        Sunny
500        2016-09-29   Month 8        Sunny
1595       2013-07-13  Month 12        Sunny
822        2015-10-23   Month 2        Sunny
1723       2013-02-04   Month 9        Sunny

Process finished with exit code 0
```

It looks like the terrestrial_date column is a string, let's change it to the UNIX timestamp to work and perform better in the algorithm.

```
# Preprocessing: Change terrestrial_date from string to datetime for optimization
dataset['terrestrial_date'] = pandas.DatetimeIndex(dataset['terrestrial_date']).astype(numpy.int64)/1000000
```

If we look at the "month" columns, it is in a standard format like "Month 5" so we are going to change it to "5" from "Month 5".

```
# Preprocessing: Change month columns from ex: "Month 5" to 5
dataset['month'] = dataset['month'].str.split(' ').str[1]
```

Since most algorithms don't work if some rows have a value of NaN, We are going to drop the rows that have a NaN value.

```
# Preprocessing: Drop rows that have NaN value
dataset.dropna(inplace=True)
```

Since we are not going to use `"id", "atmo_opacity", "wind_speed"` columns which some are NaN or unnecessary, We can remove them.

```
INEFFECTIVE_COLUMNS = ["id", "atmo_opacity", "wind_speed", ]
dataset = dataset.drop(columns=INEFFECTIVE_COLUMNS)
```

As we look through the data, we can see that some of the Integer or Float values don't have a specific range. Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. Normalization scales each input variable separately to the range 0-1, which is the range for floating-point values where we have the most precision. (Machinelearningmastery, 2020)

So first, we should choose the int or float columns for normalizing.

```
# Integer or float columns that is going to be normalized
normalized_columns = [
    "sol",
    "ls",
    "pressure"
]
```

And we can normalise them using MinMaxScaler;

```
# Preprocessing: Normalize columns for better performance in algorithms except date
normalizer = MinMaxScaler()
dataset[normalized_columns] = normalizer.fit_transform(dataset[normalized_columns])
```

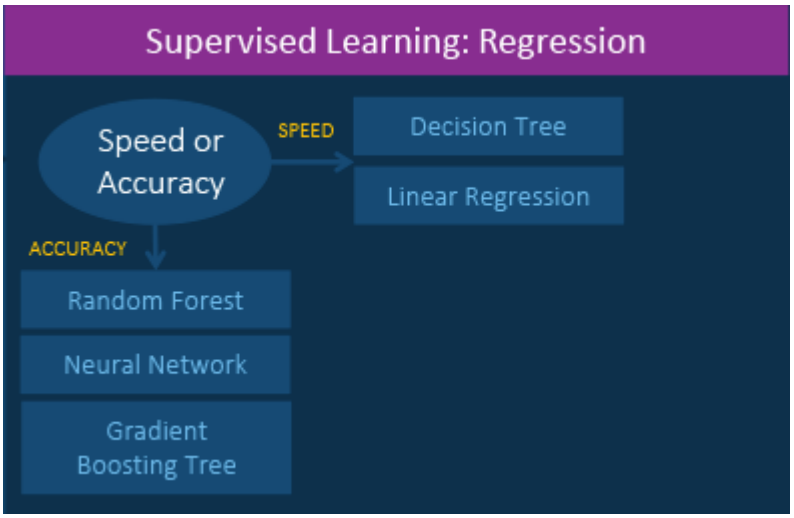
After preprocessing, We should split our data as training data and testing data to predict it. This provides us with some data to test the prediction performance of the model.

```
# Test data percentage to split from the whole dataset
TEST_PERCENTAGE = 20
```

```
# Split the data as target and training data
X = dataset.drop(columns=TARGET_COLUMNS)
Y = dataset.drop(columns=training_columns)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=TEST_PERCENTAGE / 100)
```

For the machine learning algorithm to use; since our dataset is supervised and we are aiming to predict numeric values;



These are the models that can fit our dataset. Since our dataset is below 100K sample, I aim to investigate Linear Regression, Random Forest, Decision Tree and Neural Network on our dataset and evaluate the prediction test results.

Let’s create a dictionary of models that we choose to use later on a loop.

```
TRAINING_MODELS = {
    "Neural Network": MLPRegressor(random_state=1, max_iter=10000),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "Linear Regressor": LinearRegression()
}
```

Let’s create a function to train and test the models and print the metrics such as MSE, RMSE and Cross-Validation. After that, we will execute every model through this function in the Training Models list with a loop.

```
# Train and test models then print the metrics
def train_and_test_model(model_name, model):
    model.fit(X, Y)
    prediction = model.predict(X_test)
    mse = mean_squared_error(Y_test, prediction)
    cv_score = cross_val_score(model, X, Y, cv=10)

    print("\nAlgorithm: " + model_name)
    print("MSE: %.4f" % mse)
    print("RMSE: %.4f" % mse ** (1 / 2.0))
    print("Cross validation mean: ", cv_score.mean())


# Execute train_and_test_model function for every model in the model list
for name, model in TRAINING_MODELS.items():
    train_and_test_model(name, model)
```

Analysis and Discussions

After executing the implementation above, here is the output;

```
/home/suleyman/PycharmProjects/ml-cw2/venv/bin/p

Algorithm: Neural Network
MSE: 69.3382
RMSE: 8.3270
Cross validation mean:  -7775490.6589667825

Algorithm: Decision Tree
MSE: 0.0000
RMSE: 0.0000
Cross validation mean:  -1.3866552022582808

Algorithm: Random Forest
MSE: 0.6816
RMSE: 0.8256
Cross validation mean:  -0.287498235191063

Algorithm: Linear Regressor
MSE: 36.2848
RMSE: 6.0237
Cross validation mean:  -2.3771457574839725

Process finished with exit code 0
```

A low RMSE value indicates that the simulated and observed data are close to each other showing a better accuracy. Thus lower the RMSE better the model performance. (Researchgate)

Decision Tree's MSE and RMSE value is 0. MSE being zero means our expected outputs are exactly matched by actual outputs. This can either mean the network is ideally trained and the goal is 100% or it can also mean the model is overfitting. To tackle this we also run cross-validation.

In the end, according to the output, the best performer for this dataset is the RandomForest algorithm since its MSE, RSME and CV values are better and seem to not overfit.

Github Repo link (includes full implementation):
https://github.com/suleymankalkan/ml-coursework2

Conclusions and suggestions for future work

As a result of this coursework, I improved my skills in pre-processing data to fit and optimize the algorithms, using different metrics to evaluate models.

If I have more time, I would probably fill some of the NaN values in the dataset with the average of the columns and see how it will perform. Also, I didn't have enough time to go deeper on evaluating metrics and presenting the results, If I would have more time I would add more metrics and present them in an easy to read graph and also another graph to compare the test and actual data together.

Bibliography and Citations

1. https://www.researchgate.net/publication/352976885_Mars_weather_data_analysis_using_machine_learning_techniques
2. https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/Artificial_intelligence_in_space#:~:text=NASA%20is%20also%20using%20AI,operations%20and%20space%20transportation%20systems.
3. https://www.kaggle.com/code/imkrkannan/time-series-forecasting-using-keras-cnn
4. https://www.kaggle.com/datasets/imkrkannan/mars-weather-data
5. https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/
6. https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
7. https://www.researchgate.net/post/Whats-the-acceptable-value-of-Root-Mean-Square-Error-RMSE-Sum-of-Squares-due-to-error-SSE-and-Adjusted-R-square#:~:text=A%20low%20RMSE%20value%20indicates,RMSE%20better%20is%20model%20performance.&text=The%20RMSE%20is%20a%20good,proportional%20to%20the%20observed%20mean.