Data Analysis on GSA dataset using ML algorithms

Sravanthi Pasumarthi
Computer Science
University of Houston
Houston,United States
spasumarthi@uh.edu

Sai Sri Varsha Gadde
Computer Science
University of Houston
Houston,United States
sgadde@uh.edu

Premchand Veerapalli
Computer Science
University of Houston
Houston,United States
pveerapalli@uh.edu

Abstract—

In the past decade, digitization of information has led to a data explosion in both volume and complexity. While traditional computing frameworks have failed to provide adequate computing power for the now common data-intensive computing tasks, cloud computing provides an effective alternative to enhance computing power. Machine learning algorithms are powerful analytical methods that allow machines to recognize patterns and facilitate human learning.

This research compares the accuracy, prediction of solar flares by various machine learning algorithms like Logistic Regression, Random Forests and Light GBM. We compute the confusion matrices for these three algorithms for the solar datasets (SDO, GOES).

With Amazon SageMaker, we can easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. Usage of scikit learn,tensorflow and keras helps to model, train and test the datasets. To meet the reliability and scaling needs, we can use the Amazon Simple Storage Service(S3).

Keywords— AmazonSageMaker, Machine Learning Frameworks, SDO and GOES

I. Introduction

Sun produces solar flares, which have the power to affect the Earth and near-Earth environment with their great bursts of electromagnetic energy and particles. A solar flare occurs when magnetic energy that has built up in the solar atmosphere is suddenly released. Solar flares are classified as A (smallest), B, C, M or X (strongest) according to the peak flux. The flares having highest peak flux can cause immense disruption on the nearby planets

Software systems that learn from data are being deployed in increasing numbers in industrial application scenarios. Managing these machine learning (ML) systems and the models which they apply imposes additional challenges beyond those of traditional software systems. Machine learning was once out of the reach of most enterprise budgets, but today, public cloud providers' ability to offer machine-learning services makes this technology affordable.

A select set of applications requires a storage technology that is flexible enough to let application designers configure their data store appropriately based on these tradeoffs to achieve high availability and guaranteed performance in the most cost effective manner.

AWS SageMaker is one such cloud machine learning SDK designed for speed of iteration, and it's one of the fastest-growing toys in the Amazon AWS EcoSystem. Amazon SageMaker includes the following features:
Preprocessing: Analyze and pre-process data, tackle feature engineering, and evaluate models.
Amazon Sage Maker Studio: An integrated machine learning environment where you can build, train, deploy, and analyze your models all in the same application.
SageMaker supports TensorFlow,keras,scikit-learn and many other machine learning frameworks. In this project, we have mainly used sklearn.

We need a storage service to store the data to implement the machine learning algorithms. Amazon Simple Storage Service (Amazon S3) is one such object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements.

Most of the data in the organizations is unstructured and semi-structured. Reliability at massive scale is one of the biggest challenges. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

Dataset:

The dataset that will be used for analyzation is Geostationary Operational Environmental Satellite-GOES, Solar Dynamics Observatory (SDO).

Motivation:

A flare is defined as a sudden, rapid, and intense variation in brightness. A solar flare occurs when magnetic energy that has built up in the solar atmosphere is suddenly released. Radiation is emitted across virtually the entire electromagnetic spectrum, from radio waves at the long wavelength end, through visible light to x-rays and gamma rays. The amount of energy released could power the whole world for 10 million years! On the other hand, it is less than one-tenth of the total energy emitted by the Sun every second. The first solar flare recorded in astronomical literature was on September 1, 1859.

Typically, a person cannot view a solar flare by simply staring at the Sun. Flares are in fact difficult to see because the Sun is already so bright. Instead, specialized scientific instruments are used to detect the light emitted during a flare. Radio and optical emission from flares can be observed with telescopes on Earth. Energetic emission such as x-rays and gamma-rays require telescopes located in space, since these emissions, thankfully, do not penetrate Earth's atmosphere.

Mapping between SDO and GOES:

GOES Database:

- Consider flares with a Geostationary Operational Environmental Satellite (GOES) X-ray flux peak magnitude above the M1.0 level (major flares)
- Positive event to be an active region that flares with a peak magnitude above the M1.0 level, as defined by the GOES database.
- A negative event would be an active region that does not have such an event within a 24- hour time span.
- For collection active region for the negative class, we will also use information about all regions where X-ray flux peak magnitude above the C1.0 level.

SDO Database:

The Solar Dynamic's Observatory's Helio-seismic and Magnetic Imager (HMI) instrument used to continuously map the vector magnetic field of the sun. Using this data, we can characterize active regions on the sun. The SHARP data series provide maps in patches that encompass automatically tracked magnetic concentrations for their entire lifetime. We focused on 18 parameters calculated using the SHARP vector magnetic field data. Unsigned flux, Mean shear angle, Mean current helicity, Gradient of total field, Gradient of vertical field, Gradient of horizontal field etc.

II. Approach

Our goal is to analyze GSA dataset by deploying it on the cloud and apply various machine learning algorithms to perform analysis and predict whose accuracy is better.

Dataset Description:

Classification of Solar Flares:

- Solar flares are classified as A, B, C, M or X according to the peak flux (in watts per square metre, W/m2) of 1 to 8 Angstroms X-rays near Earth, as measured by XRS instrument on-board the GOES-15 satellite which is in a geostationary orbit over the Pacific Ocean. A&B class: The A & B-class are the lowest class of solar flares. C class: C-class solar flares are minor solar flares that have little to no effect on Earth.
- M Class: M-class solar flares are the medium large solar flares. They cause small to moderate radio blackouts.
- X Class: X-class solar flares are the biggest and strongest of them all. Strong to extreme radio blackouts occur on the daylight side of the Earth during the solar flare.

Firstly, we will upload the GSA dataset csv files into the Amazon Simple Storage Services(S3) buckets which is designed to make web scale computing easier and store and retrieve any amount of data at anytime from anywhere on the web.

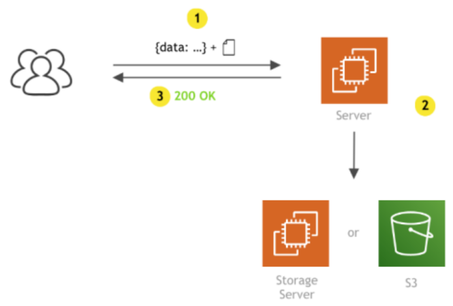


Fig 1: Storing data into a S3 bucket.

With the data present in S3 bucket, we apply various machine learning models Logistic Regression and Random Forests.

Logistic regression: It is basically a supervised classification algorithm. In a classification problem, we analyze a dataset in which there are one or more independent variables that determine an outcome. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Random forest is a supervised learning algorithm which is used for both classification as well as regression. Random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting.

Light GBM: It is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

We can implement the above algorithms using Amazon SageMaker.In machine learning, you "teach" a computer to make predictions, or inferences. First, you use an algorithm and example data to train a model. Then you integrate your model into your application to generate inferences in real time and at scale. In a production environment, a model typically learns from millions of example data items and produces inferences in hundreds to less than 20 milliseconds. The following diagram illustrates the typical workflow for creating a machine learning model:

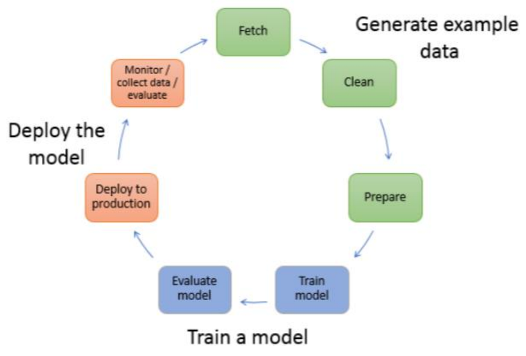


Fig 3: Training a data model.

As the diagram illustrates, we typically perform the following activities.

1) To preprocess data, we typically do the following:

   a. Fetch the data

   b. Clean the data

   c. Prepare or transform the data

In Amazon SageMaker, we preprocess data in a Jupyter notebook on your notebook instance. We use notebook to fetch your dataset, explore it, and prepare it for model training.

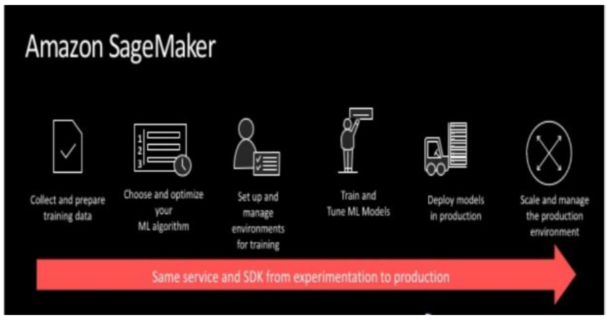
2) Train a model—Model training includes both training and evaluating the model, as follows:

a) Training the model: To train a model, you need an algorithm. We can use the previously specified algorithms or one of the algorithms that Amazon SageMaker provides. Depending on the size of your training dataset and how quickly you need the results, you can use resources ranging from a single general-purpose instance to a distributed cluster of GPU instances.

b) Evaluating the model: After we have trained the model, evaluate it to determine whether the accuracy of the inferences is acceptable. In Amazon SageMaker, we use either the AWS SDK for Python (Boto) or the high-level Python library that Amazon SageMaker provides to send requests to the model for inferences. We can also use a Jupyter notebook in our Amazon SageMaker notebook instance to train and evaluate the model.

3) Deploy the model— We traditionally re-engineer a model before integrating it with the application and deploy it. With Amazon SageMaker hosting services, we can deploy the model independently, decoupling it from the application code.

4) Validating the model-After training a model, evaluate the trained model to determine its performance and accuracy. We can generate multiple models using different methods and evaluate each of it. Validating Models measures to determine each model's suitability.

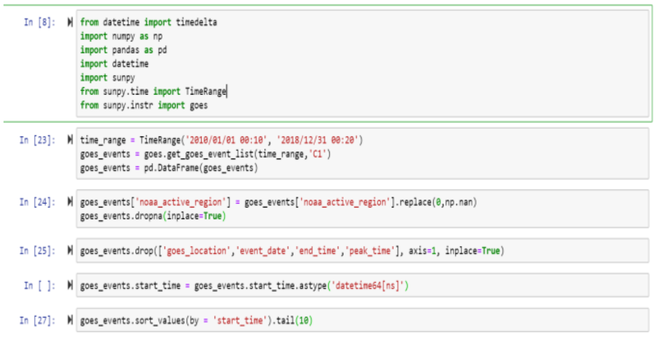

Amazon SageMaker
Collect and prepare training data
Choose and optimize your ML algorithm
Set up and manage environments for training
Train and Tune ML Models
Deploy models in production
Scale and manage the production environment
Same service and SDK from experimentation to production

III Implementation: Our Implemenatation is as follows:

1. Data sources:

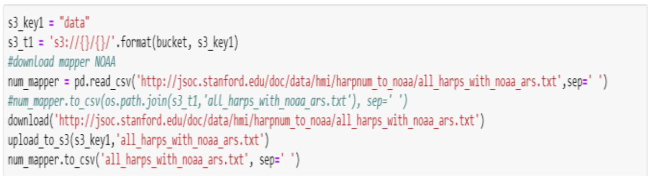
a. Downloading dataset in CSV format:

Here data is collected from two instruments: GOES and SDO. For handling Solar data there is special package “sunpy” in python.



```
In [8]:   from datetime import timedelta
          import numpy as np
          import pandas as pd
          import datetime
          import sunpy
          from sunpy.time import TimeRange
          from sunpy.instr import goes

In [23]:  time_range = TimeRange('2010/01/01 00:10', '2018/12/31 00:20')
          goes_events = goes.get_goes_event_list(time_range,'C1')
          goes_events = pd.DataFrame(goes_events)

In [24]:  goes_events['noaa_active_region'] = goes_events['noaa_active_region'].replace(0,np.nan)
          goes_events.dropna(inplace=True)

In [25]:  goes_events.drop(['goes_location','event_date','end_time','peak_time'], axis=1, inplace=True)

In [ ]:   goes_events.start_time = goes_events.start_time.astype('datetime64[ns]')

In [27]:  goes_events.sort_values(by = 'start_time').tail(10)
```

b. Uploading a complete dataset to AWS S3 Storage.



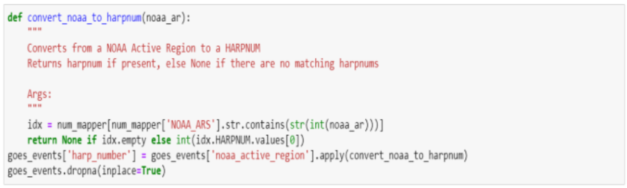
```
s3_key1 = "data"
s3_t1 = 's3://{}/{}/'.format(bucket, s3_key1)
#download mapper NOAA
num_mapper = pd.read_csv('http://jsoc.stanford.edu/doc/data/hmi/harpnum_to_noaa/all_harps_with_noaa_ars.txt',sep=' ')
#num_mapper.to_csv(os.path.join(s3_t1,'all_harps_with_noaa_ars.txt'), sep=' ')
download('http://jsoc.stanford.edu/doc/data/hmi/harpnum_to_noaa/all_harps_with_noaa_ars.txt')
upload_to_s3(s3_key1,'all_harps_with_noaa_ars.txt')
num_mapper.to_csv('all_harps_with_noaa_ars.txt', sep=' ')
```

2. Data preprocessing:

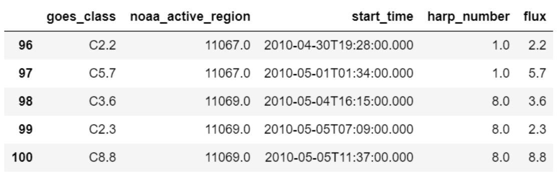
a. Fetch the data— Fetch the data from AWS S3.

b. Clean the data— The raw input data must be preprocessed and can’t be used directly for making predictions. This is because most ML models expect the data in a predefined format, so the raw data needs to be first cleaned and formatted in order for the ML model to process the data. There are a large number of different characteristics that can be used for magnetic field complexity description. For HMI/SDO magnetograms, automated active region tracking exist known as SHARP (Space weather HMI Active Region Patch). For each active region, key features known as SHARP parameters were calculated. Calculation of these features are based on SDO magnetograms.

Active region detection: Different approaches are present to detect active regions. One of them is done by NOAA, correcting manually each day. Active regions have NOAA numbers. SDO has automated system for AR detections, the regions are known as HARPs. Also, SDO compute plenty of parameters of magnetic field complexity. So harp regions with features are used, but information about goes flux there is only for NOAA regions. There is no coincidence between HARP and NOAA regions, but they can be mapped. Below the code for mapping between the HARP and NOAA regions.

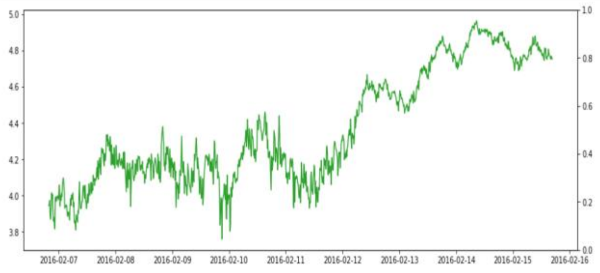


```
def convert_noaa_to_harpnum(noaa_ar):
    """
    Converts from a NOAA Active Region to a HARPNUM
    Returns harpnum if present, else None if there are no matching harpnums

    Args:
    """
    idx = num_mapper[num_mapper['NOAA_ARS'].str.contains(str(int(noaa_ar)))]
    return None if idx.empty else int(idx.HARPNUM.values[0])
goes_events['harp_number'] = goes_events['noaa_active_region'].apply(convert_noaa_to_harpnum)
goes_events.dropna(inplace=True)
```

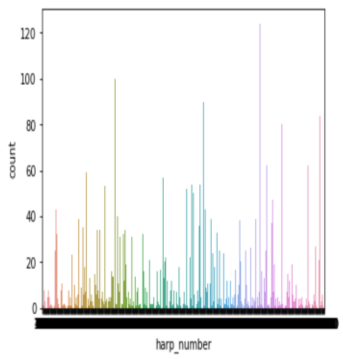


	goes_class	noaa_active_region	start_time	harp_number	flux
96	C2.2	11067.0	2010-04-30T19:28:00.000	1.0	2.2
97	C5.7	11067.0	2010-05-01T01:34:00.000	1.0	5.7
98	C3.6	11069.0	2010-05-04T16:15:00.000	8.0	3.6
99	C2.3	11069.0	2010-05-05T07:09:00.000	8.0	2.3
100	C8.8	11069.0	2010-05-05T11:37:00.000	8.0	8.8

Change in flux value for a specific harp number 6327 for 2-hour span on a given day.

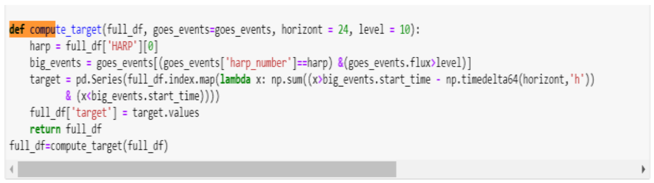


```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7d87cce8d0>
```



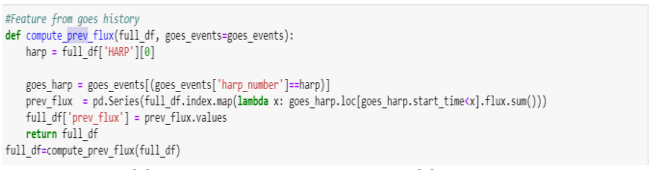
The above graph shows the plot for count vs harp number

Defining target: Positive event to be an active region that flares with a peak magnitude above the M1.0 level, as defined by the GOES database. A negative event would be an active region that does not have such an event within a 24-hour time span.



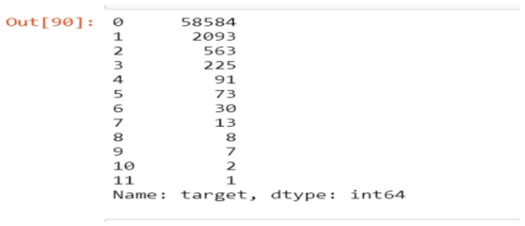
```
def compute_target(full_df, goes_events=goes_events, horizont = 24, level = 10):
    harp = full_df['HARP'][0]
    big_events = goes_events[(goes_events['harp_number']==harp) &(goes_events.flux>level)]
    target = pd.Series(full_df.index.map(lambda x: np.sum((x>big_events.start_time - np.timedelta64(horizont,'h'))
            & (x<big_events.start_time))))
    full_df['target'] = target.values
    return full_df
full_df=compute_target(full_df)
```

Computing the previous flux: Previous flux is calculated up to considered moment.



```
#Feature from goes history
def compute_prev_flux(full_df, goes_events=goes_events):
    harp = full_df['HARP'][0]

    goes_harp = goes_events[(goes_events['harp_number']==harp)]
    prev_flux  = pd.Series(full_df.index.map(lambda x: goes_harp.loc[goes_harp.start_time<x].flux.sum()))
    full_df['prev_flux'] = prev_flux.values
    return full_df
full_df=compute_prev_flux(full_df)
```

As strength of flare increases, the number of flares decreases implies strongest flares are very rare.



```
Out[90]:  0        58584
          1         2093
          2          563
          3          225
          4           91
          5           73
          6           30
          7           13
          8            8
          9            7
          10           2
          11           1
          Name: target, dtype: int64
```

C. Prepare or transform the data: We have split the data into 3 sets.

A training set consisting of 80% of the original data. Training an accurate machine learning (ML) model requires many different steps, but none is potentially more important than preprocessing your data set which is done in the previous step.

The test data (remaining 20% of data) will be used to evaluate the performance of the model, and measure how well the trained model generalizes to unseen data.

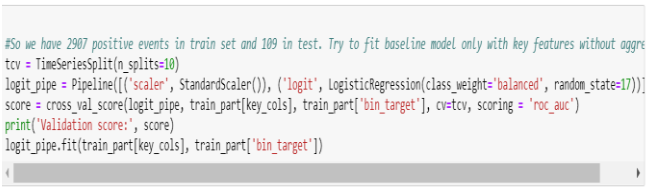
Validation set for the algorithms to perform the proper evaluation of the model.

ML algorithms:

1) Exporting data to the Python environment Jupyter notebook.

2) ML models creation and configuration: We have implemented machine learning models Logistic Regression and Random Forests on the dataset and performed data analysis.

Logistic Regression:



```
#So we have 2907 positive events in train set and 109 in test. Try to fit baseline model only with key features without aggre
tcv = TimeSeriesSplit(n_splits=10)
logit_pipe = Pipeline([('scaler', StandardScaler()), ('logit', LogisticRegression(class_weight='balanced', random_state=17))]
score = cross_val_score(logit_pipe, train_part[key_cols], train_part['bin_target'], cv=tcv, scoring = 'roc_auc')
print('Validation score:', score)
logit_pipe.fit(train_part[key_cols], train_part['bin_target'])
```

Confusion Matrix: A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.

The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

To calculate a Confusion Matrix

Below is the process for calculating a confusion Matrix.

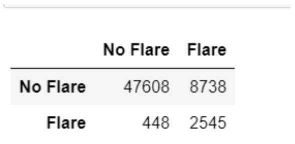
1. You need a test dataset or a validation dataset with expected outcome values.

2. Make a prediction for each row in your test dataset.

3. From the expected outcomes and predictions count:

- The number of correct predictions for each class.
- The number of incorrect predictions for each class, organized by the class that was predicted.

Results on Training data for logistic Regression:

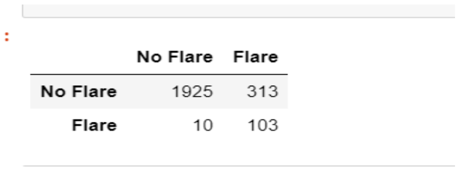


	No Flare	Flare
No Flare	47608	8738
Flare	448	2545

True positives: 2545 False Positives: 8738

True negatives: 47608 False negatives: 448

Results on Testing Data for Logistic Regression:



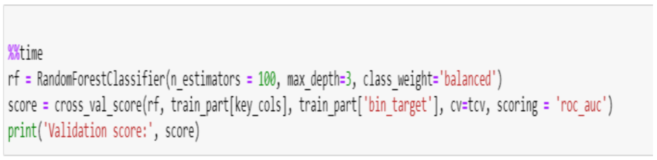
	No Flare	Flare
No Flare	1925	313
Flare	10	103

True positives: 103 False Positives: 313

True negatives: 1925 False negatives: 10

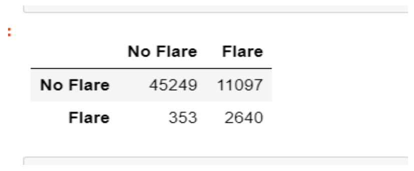
From confusion matrix, we identify the number of correct predictions and incorrect predictions made by the algorithm and calculate the accuracy.
Accuracy for logistic regression is 86.26% and error rate is 13.78%

Random Forest:



```
%%time
rf = RandomForestClassifier(n_estimators = 100, max_depth=3, class_weight='balanced')
score = cross_val_score(rf, train_part[key_cols], train_part['bin_target'], cv=tcv, scoring = 'roc_auc')
print('Validation score:', score)
```

Results on Training data for Random Forest:



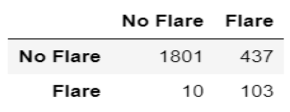
:

	No Flare	Flare
No Flare	45249	11097
Flare	353	2640

True positives: 2640 False Positives: 11097

True negatives: 353 False negatives: 45249

Results on Testing Data for Random Forest:

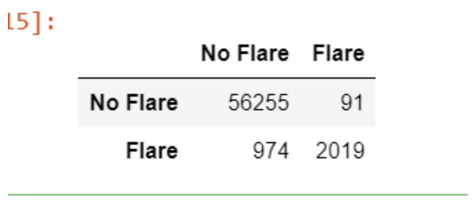


	No Flare	Flare
No Flare	1801	437
Flare	10	103

True positives: 103 False Positives: 437

True negatives: 1801 False negatives:10

Accuracy for Random Forest is 80.98% and error rate is 19.02%
Inference:
Results on Training data for Light GBM:

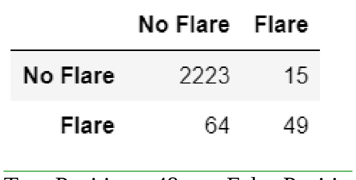


[5]:

	No Flare	Flare
No Flare	56255	91
Flare	974	2019

True Positives: 2019 False Positives: 91
True Negatives: 56255 False Negatives: 974

Results on Testing data for Light GBM:



	No Flare	Flare
No Flare	2223	15
Flare	64	49

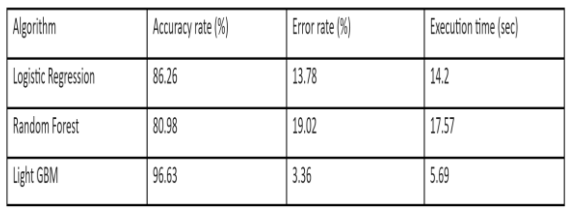
True Positives: 49 False Positives: 15
True Negatives: 2223 False Negatives: 64

Accuracy for Light BGM is 96.63% and error rate is 3.36.

Results:

Logistic regression works better than random forests with an accuracy of 86.26%.

Light GBM works really fast with an accuracy of 96.63%.



Algorithm	Accuracy rate (%)	Error rate (%)	Execution time (sec)
Logistic Regression	86.26	13.78	14.2
Random Forest	80.98	19.02	17.57
Light GBM	96.63	3.36	5.69

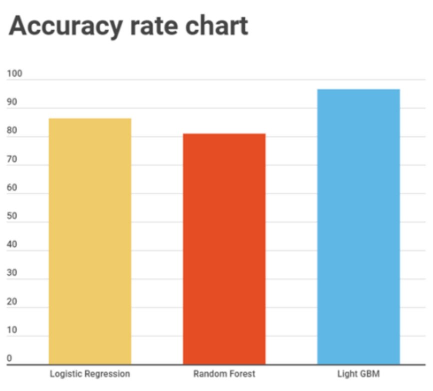


Fig- Bar chart inferring three different algorithms accuracy

For all the three algorithms we have plotted execution time for both training and testing data.

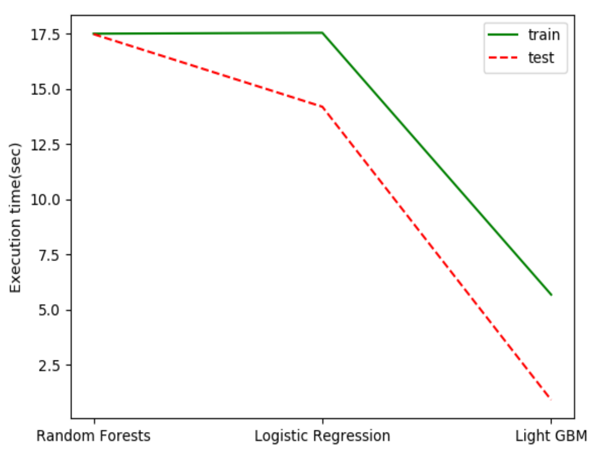


Fig- Execution times for both Training and Testing data for all three algorithms

Cloud Services:

1. As our dataset has huge amount of data, with Amazon S3 it became easy to store and access as much data (individual objects can only be up to 5 terabytes in size) you want and whenever you want to and also there is no need to predict future data usage.
2. Initially we thought of using Dynamo Db by exporting data from S3, After that we realized using csv file from S3 bucket would be more compatible. As Amazon S3 automatically creates multiple replicas of your data so that our data is never lost. It became really easy to recovery the data as S3 provides different versions of data.
3. Amazon Sagemaker enabled feeding training data directly from Amazon S3, removing any storage space constraints.
4. We have used notebook instance to collect and prepare data to define a model, and to start the learning process. We have used python, popular ML frameworks and SageMaker's own libraries.

Tools Used:

Machine Learning Framework	TensorFlow, keras, SciKit-learn, PyTorch
Machine Learning Cloud Service	Amazon SageMaker
Storage Service	Amazon S3(Amazon Simple **Storage** Service)
Notebook Instance	Jupyter Notebook

Challenges Faced:

1. It was hard to combine the datasets given by GOES and SDO based on their harp numbers.
2. It was difficult to filter the sun related data based on the SHARP features from a huge volume of available data.
3. Identifying the appropriate inbuilt functions of drms, sunpy modules to apply on the time series data was a challenging task.
4. we couldn't input the data from Dynamo DB to ML algorithms. Hence we reverted to taking the data from S3 bucket.

Conclusion and Future Work:

In this report, we detailed 3 Machine Learning algorithms to predict solar flares. A dataset was compiled by the data gathered over a span of 9 years from SDO, GOES satellites developed by NASA. We first identified the active regions using the NOAA numbers and then mapped the data in SDO using harp numbers.
We have considered 18 crucial features identified by SHARP for analyzing the occurrence of solar flares. After identifying the active regions considering the flux values, we have created a mapping on how these SHARP features vary according to a 2-hour span on a given day. We have provided a heat map for the same considering all the 18 features of SHARP. We have understood that logistic regression works better than random forests with an accuracy of 86.26%.

Light GBM works really fast with an accuracy of 96.63%. In future we can more analyze and predict whether these really harm the earth's atmosphere by taking time series data into consideration.

IV Roles:

Responsibility of Varsha:

- Applied Random forests, plotted accuracy and confusion matrices for the train and test data.
- Extracted feature related data from the combined dataset and sort it into csv files based on the harp numbers.

V **References:**

[1] (PDF) Integration of NoSQL Databases for Analyzing Spatial Information in Geographic Information System
[2] Amazon SageMaker - Developer Guide
[3] (PDF) Practical machine learning based on cloud computing resources
[5] https://en.wikipedia.org/wiki/List_of_GOES_satellites
[4] http://jsoc.stanford.edu/new/HMI/HARPS.html
[5] https://sdo.gsfc.nasa.gov/
[6] http://hmi.stanford.edu/magnetic/
[7] http://jsoc.stanford.edu/jsocwiki/HARPDataSeries
[8] https://en.wikipedia.org/wiki/Stereographic_projection
[9] https://en.wikipedia.org/wiki/Solar_flare#Classification
[10] https://docs.sunpy.org/projects/drms/en/latest/
[11] http://docs.sunpy.org/en/latest/generated/gallery/index.html#acquiring-data