Project Report

CCSC: Solardb

https://nature.njit.edu/solardb/ccsc

Masters Project: CS-700 B FALL-22

Professor: Jason T Wang

Department of Computer Science

Kavya Basavalingappa Umashankar

UCID – kbu2

Contents

1. Introduction

1.1 Solar Flares

A solar flare is an intense localized eruption of electromagnetic radiation in the Sun's atmosphere. Flares occur in active regions and are often, but not always, accompanied by coronal mass ejections, solar particle events, and other solar phenomena. The occurrence of solar flares varies with the 11-year solar cycle.

Solar flares are thought to occur when stored magnetic energy in the Sun's atmosphere accelerates charged particles in the surrounding plasma. This results in the emission of electromagnetic radiation across the electromagnetic spectrum.

High-energy electromagnetic radiation from solar flares is absorbed by the daylight side of Earth's upper atmosphere, in particular the ionosphere, and does not reach the surface. This absorption can temporarily increase the ionization of the ionosphere which may interfere with short-wave radio communication. The prediction of solar flares is an active area of research.

1.2 Machine Learning

Machine learning (ML) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, agriculture, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

1.3 Deep learning

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks and Transformers have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, climate science, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

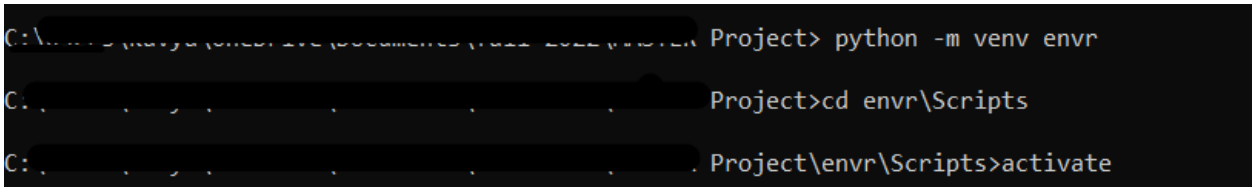
2. Requirements and Setting up

2.1 Setting up using Windows Command prompt

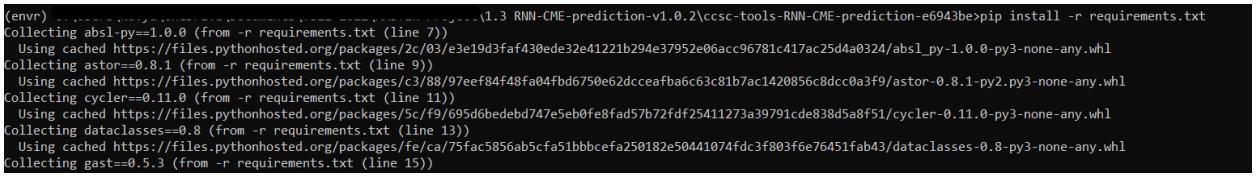
Each tool requires a set up for it to execute smoothly. In this section, some of the basic requirements and set up for all the tools are mentioned step by step. We can download the tools from zenodo and set up the environment.

Each of these project require different python versions to run, some of them are python 3.6.x, python 3.8.x, python 3.9.x. Install the required python version from https://www.python.org/downloads/ . After downloading, we can run the installation package.

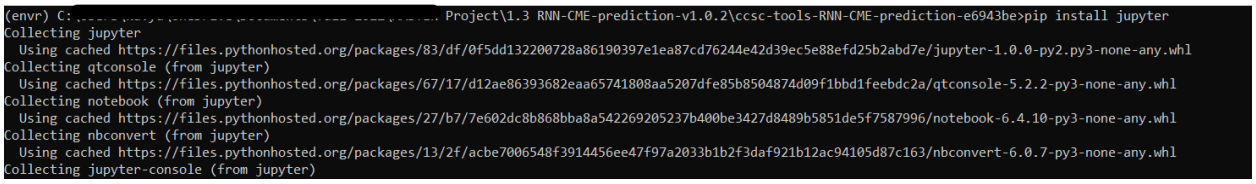
Next step is to create an virtual environment to install all the required package. In command prompt we can navigate to the desidered directory and the command for creating virtual environment: **python -m venv name_of_virtualenv**. If the name of virtual environment is "envr", then we can navigate to envr\Scripts in command prompt and run activate to activate the virtual environment.



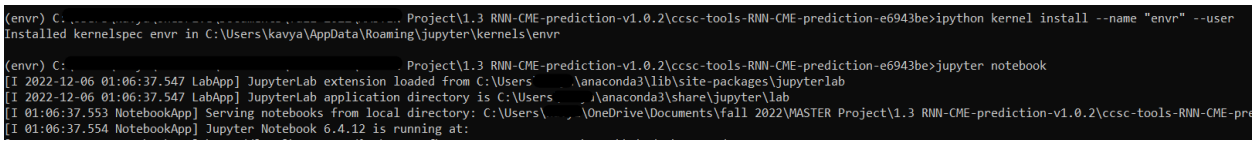
We will install all the required modules from requirements.txt by running the following command: **pip install -r requirements.txt**.



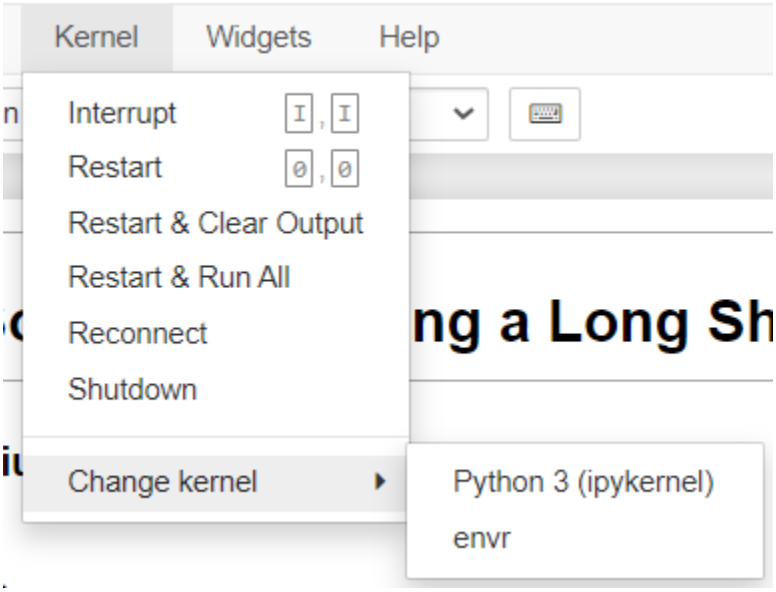
Next, to install jupyter notebook in this environment by : **pip install jupyter notebook**.



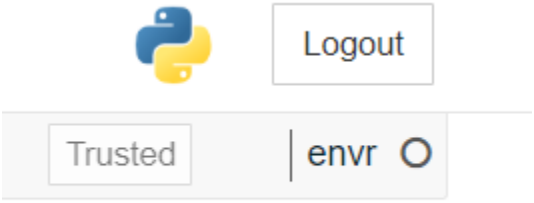
To add this virtual envoirnment kernel to the jupyter notebook : **ipython kernel install --name "envr" --user**. We can launch jupyter notebook by typing in **jupyter notebook** in command prompt.



We can open the notebook and make sure the kernel that is running is the virtual environment kernel by changing the kernel to envr.



We can confirm the kernel which is used to run the notebook:

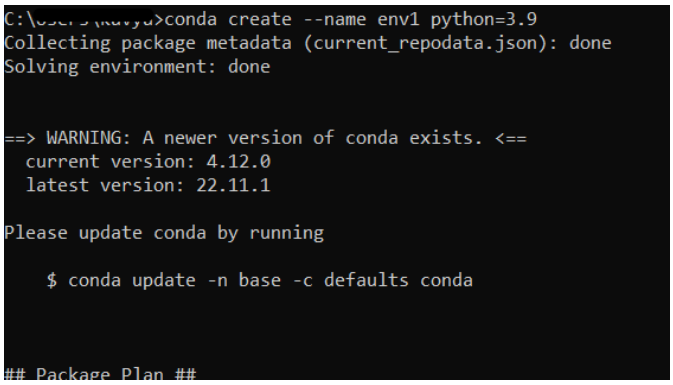


We can remove the unwanted kernel by **jupyter kernelspec uninstall unwanted-kernelname** in the command prompt.

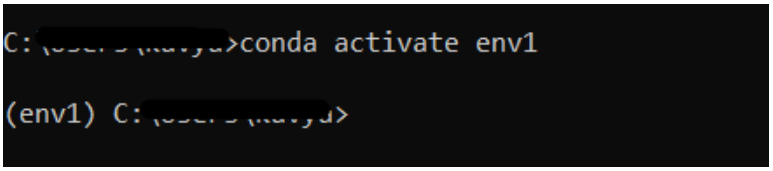
2.2 Setting up using Miniconda

We have different versions of Miniconda for different python versions available on https://docs.conda.io/en/latest/miniconda.html. Install the required version for the tool and set up the installation.

We can use windows command prompt terminal or the miniconda terminal to set up an environment. To create the environment: **conda create --name env1 python=3.X** where env1 is the environment name.



To activate the environment: **conda activate env1**



If there is need to set up gpu, then we can **conda install -c anaconda tensorflow-gpu**

```
(env1) C:\Users\kavya>conda install -c anaconda tensorflow-gpu
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
```

We can navigate to the folder location and install the requirements from the text file in two ways:

- Using conda command: **conda install --file requirements.txt**

```
(env1) C:\Users\kavya\OneDrive\Documents\fall 2022\MASTER Project\1.8 Kp-prediction-v1.1\ccsc-tools-Kp-prediction-4b
06c9d>conda install --file requirements.txt
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Collecting package metadata (repodata.json): done
```

- Using pip:
    - We install pip in miniconda and then install requirements through pip:
    - Install pip: **conda install pip**

```
(env1) C:\Users\kavya\OneDrive\Documents\fall 2022\MASTER Project\1.8 Kp-prediction-v1.1\ccsc-tools-Kp-prediction-4b06c9
d>conda install pip
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

    - Install requirements: **pip install -r requirements.txt**

```
(env1) C:\Users\kavya\OneDrive\Documents\fall 2022\MASTER Project\1.8 Kp-prediction-v1.1\ccsc-tools-Kp-prediction-4b
06c9d>pip install -r requirements.txt
Collecting tensorflow==2.8.0
  Using cached tensorflow-2.8.0-cp39-cp39-win_amd64.whl (438.0 MB)
Collecting tensorflow-gpu==2.8.0
```

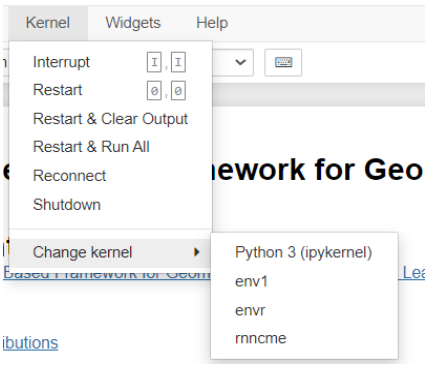
Next, we need to install the ipykernel: **conda install -c anaconda ipykernel**

```
(env1) C:\Users\kavya\OneDrive\Documents\fall 2022\MASTER Project\1.8 Kp-prediction-v1.1\ccsc-tools-Kp-prediction-4b
06c9d>conda install -c anaconda ipykernel
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Add kernel to jupyter: **python -m ipykernel –user –name=env1**

```
(env1) C:\Users\kavya\OneDrive\Documents\fall 2022\MASTER Project\1.8 Kp-prediction-v1.1\ccsc-tools-Kp-prediction-4b
06c9d>python -m ipykernel install --user --name=env1
Installed kernelspec env1 in C:\Users\kavya\AppData\Roaming\jupyter\kernels\env1
```

We can open the notebook: **jupyter notebook** and make sure the kernel that is running is the virtual environment kernel by changing the kernel to the env1.



3. Tools

3.1 Identifying and Tracking Solar Magnetic Flux Elements with Deep Learning

Tool Version-1.2: A new deep learning method, called SolarUnet, to identify and track solar magnetic flux elements or features in observed vector magnetograms. To run this project we can download python version: 3.8.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

- Data Preperation:

2.1 Data Preparation

Import the pre_processing() function from the solarunet module.

Convert the SWAMIS 3-class masks to 2-class masks for model training. You may put your data into this directory.

In [1]:
```
from solarunet import pre_processing
input_path = 'data/exmaple_data_preprocess/3_class/'
output_path = 'data/exmaple_data_preprocess/2_class/'
print('Pre-processing example data...')
pre_processing(input_path, output_path)
print('Finished the pre-processing.')
```

```
Pre-processing example data...
intgr_180607_161752.fts
intgr_180607_161904.fts
Finished the pre-processing.
```

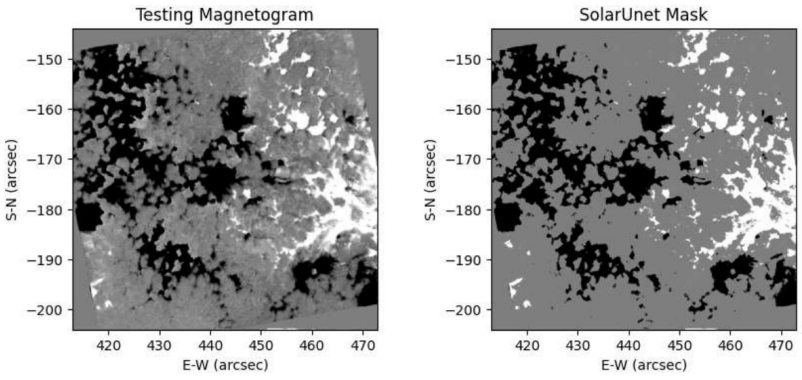
- Predicting with the pretrained model:

In [2]:
```
from solarunet import model_predicting
input_path ='data/magnetic/'
output_path ='results/predicted_mask/'
model_predicting(input_path, output_path, pretrain=True)
```

```
3/3 [==============================] - 2s 568ms/step
Prediction on the given data done
```

- Postprocessing Data

In [3]:
```
from solarunet import post_processing
from solarunet import plot_mask
%matplotlib inline
post_processing()
plot_mask()
```



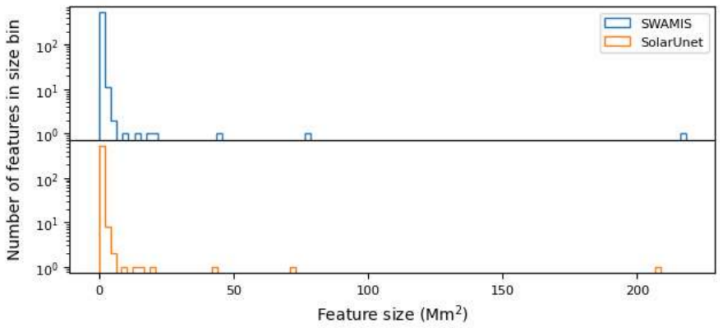
- Magnetic Tracking

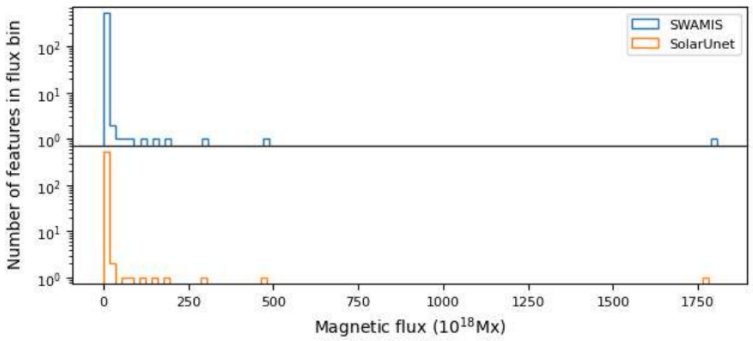
In [4]:
```
from magnetic_tracking import magnetic_tracking
input_path = 'results/processed_data_for_tracking/'
output_path = 'results/tracking_results/'
magnetic_tracking(input_path, output_path)
# lifetime_path = 'data/statistics_analysis/lifetime'
# magnetic_tracking(input_path, output_path, lifetime_path)
```

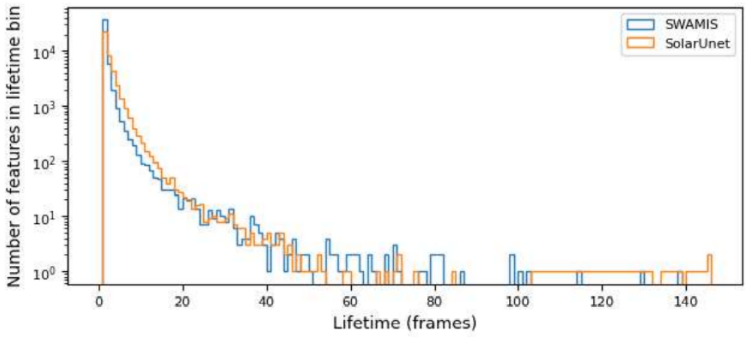
```
============magnetic tracking start============
-----------------process frame 1------------------
-----------------process frame 2------------------
-----------------process frame 3------------------
-----------------Done------------------
```

- Statistics Analysis

In [6]:
```
from statistics_analysis import analysis
%matplotlib inline
analysis()
```







3.2 Predicting Solar Flares with Machine Learning

the solar flare prediction problem at hand is essentially a multi-class (B, C, M, X) classification problem. The FlareML system employs four machine learning methods to tackle this multi-class prediction problem. These four methods are: (i) ensemble (ENS), (ii) random forests (RF), (iii) multilayer perceptrons (MLP), and (iv) extreme learning machines (ELM).

Tool Version-1.3: To run this project we can download python version: 3.9.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

- ENS Model Training and Testing
    - ENS Model Training with Default Data

In [3]:
```
print('Loading the train_model function...')
from flareml_train import train_model
args = {'train_data_file':'data/train_data/flaringar_training_sample.csv',
        'algorithm': 'ENS',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: dat
a/train_data/flaringar_training_sample.csv
Loading data set...
Training is in progress, please wait until it is done...
Finished 1/3 training..
Finished 2/3 training..
Finished 3/3 training..

Finished training the ENS model, you may use the flareml_test.py program to mak
e prediction.
```

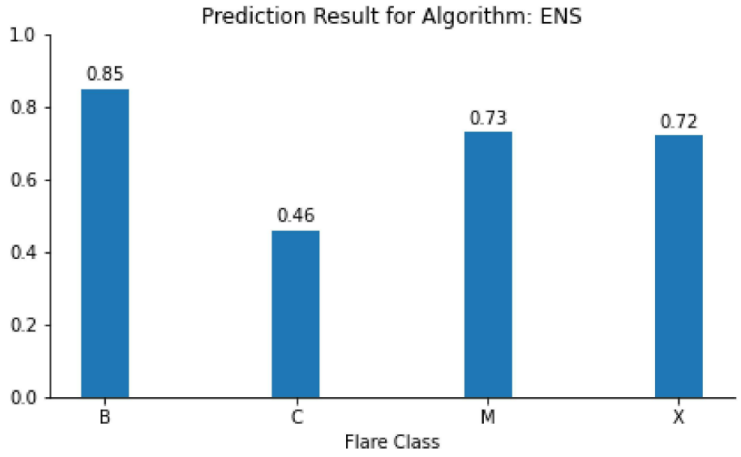
    - Predicting with Your ENS Model

In [4]:
```
from flareml_test import test_model
args =  {'test_data_file': 'data/test_data/flaringar_simple_random_40.csv',
         'algorithm': 'ENS',
         'modelid': 'custom_model_id'}
custom_result = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/
test_data/flaringar_simple_random_40.csv
Loading data set...
Done loading data...
Formatting and mapping the flares classes..
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

- Plotting the Results

In [5]:
```
from flareml_utils import plot_custom_result
plot_custom_result(custom_result)
```



- RF Model Training and Testing
  - RF Model Training with Default Data

In [6]:
```
print('Loading the train_model function...')
from flareml_train import train_model
args = {'train_data_file':'data/train_data/flaringar_training_sample.csv',
        'algorithm': 'RF',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: dat
a/train_data/flaringar_training_sample.csv
Loading data set...
Training is in progress, please wait until it is done...

Finished training the RF model, you may use the flareml_test.py program to make
prediction.
```

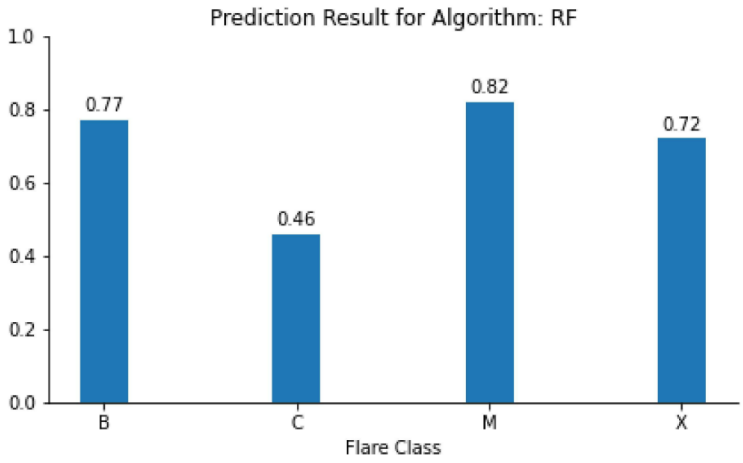
- Predicting with Your RF Model

In [7]:
```
from flareml_test import test_model
args =  {'test_data_file': 'data/test_data/flaringar_simple_random_40.csv',
         'algorithm': 'RF',
         'modelid': 'custom_model_id'}
custom_result = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/
test_data/flaringar_simple_random_40.csv
Loading data set...
Done loading data...
Formatting and mapping the flares classes..
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

- Plotting the Results

In [8]:
```
from flareml_utils import plot_custom_result
plot_custom_result(custom_result)
```



- MLP Model Training and Testing
  - MLP Model Training with Default Data

In [9]:
```
print('Loading the train_model function...')
from flareml_train import train_model
args = {'train_data_file':'data/train_data/flaringar_training_sample.csv',
        'algorithm': 'MLP',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: dat
a/train_data/flaringar_training_sample.csv
Loading data set...
Training is in progress, please wait until it is done...

Finished training the MLP model, you may use the flareml_test.py program to mak
e prediction.
```

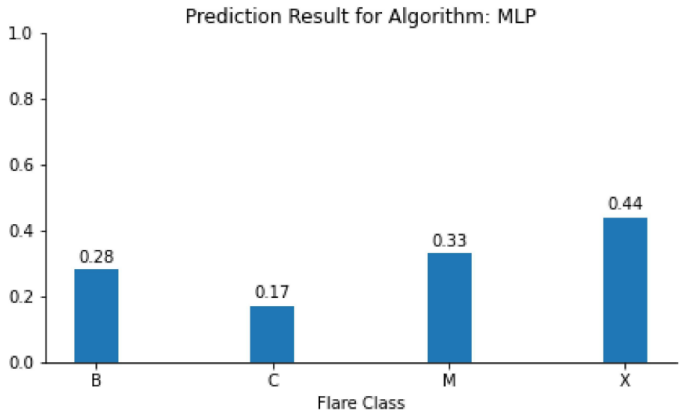
  - Predicting with Your MLP Model

In [10]:
```
from flareml_test import test_model
args =  {'test_data_file': 'data/test_data/flaringar_simple_random_40.csv',
         'algorithm': 'MLP',
         'modelid': 'custom_model_id'}
custom_result = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/
test_data/flaringar_simple_random_40.csv
Loading data set...
Done loading data...
Formatting and mapping the flares classes..
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

  - Plotting the Results

In [11]:
```
from flareml_utils import plot_custom_result
plot_custom_result(custom_result)
```



- ELM Model Training and Testing
  - ELM Model Training with Default Data

In [12]:
```
print('Loading the train_model function...')
from flareml_train import train_model
args = {'train_data_file':'data/train_data/flaringar_training_sample.csv',
        'algorithm': 'ELM',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: dat
a/train_data/flaringar_training_sample.csv
Loading data set...
Training is in progress, please wait until it is done...

Finished training the ELM model, you may use the flareml_test.py program to mak
e prediction.
```

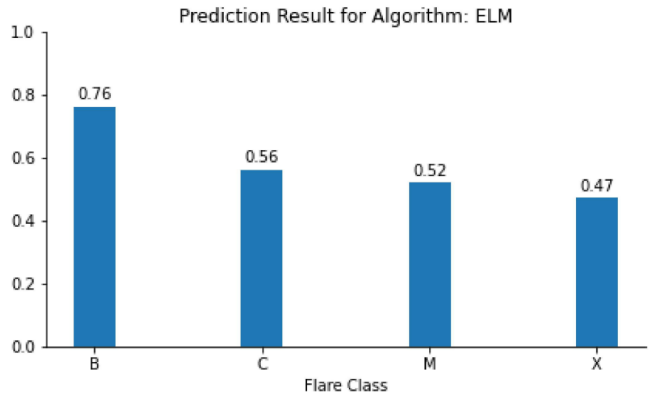
  - Predicting with Your ELM Model

In [13]:
```
from flareml_test import test_model
args =  {'test_data_file': 'data/test_data/flaringar_simple_random_40.csv',
         'algorithm': 'ELM',
         'modelid': 'custom_model_id'}
custom_result = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/
test_data/flaringar_simple_random_40.csv
Loading data set...
Done loading data...
Formatting and mapping the flares classes..
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

  - Plotting the Results

In [14]:
```
from flareml_utils import plot_custom_result
plot_custom_result(custom_result)
```



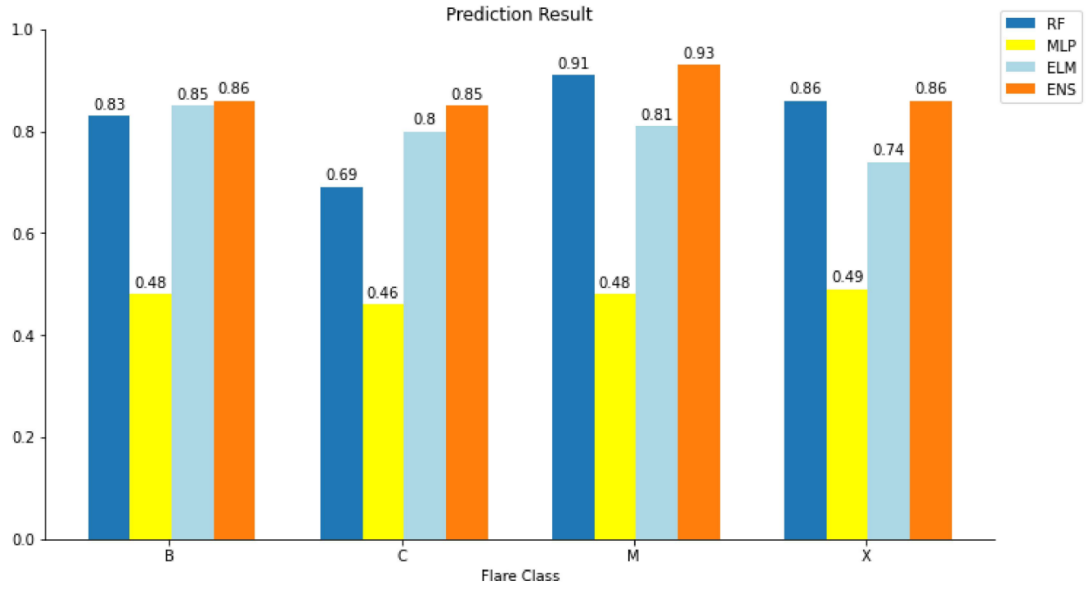
- Predicting with Pretrained Models

In [15]:
```
from flareml_test import test_model
args =  {'test_data_file': 'data/test_data/flaringar_simple_random_40.csv',
         'modelid': 'default_model'}
result = test_model(args)
```

```
Starting testing with a model with id: default_model testing data file: data/te
st_data/flaringar_simple_random_40.csv
Loading data set...
Done loading data...
Formatting and mapping the flares classes..
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

  - Plotting the Results

In [16]:
```
from flareml_utils import plot_result
plot_result(result)
```



3.3 Predicting Coronal Mass Ejections Using SDO/HMI Vector Magnetic Data Products and Recurrent Neural Networks

we demonstrate 2 machine learning models to predict whether an AR that produces an M- or X-class flare will also produce a CME. The machine learning algorithms which we use include two types of recurrent neural networks (RNNs): a long short-term memory (LSTM) network and a gated recurrent unit (GRU) network.

Tool Version-1.3: To run this project we can download python version: 3.6.x, and set up the environment either in virtual environment, install the required modules. Launch jupyter notebook and run the cells. Make sure the protobuf version is 3.20.x or lower.

- Configuration

In [1]:
```
import sys
sys.path.insert(0, './CMEpredict')

from CMEpredict import *

import pandas as pd
import matplotlib.pyplot as plt
from cycler import cycler
from sklearn.metrics import confusion_matrix as cm, roc_curve, roc_auc_score
```

In [2]:
```
mask_value = 0
series_len = 20
epochs = 20
batch_size = 256
nclass = 2
start_feature = 4
```

- Data Explanation and Sampling

In [3]:
```
names = ['Label', 'Timestamp', 'NOAA AR NUM', 'HARP NUM',
         'TOTUSJH', 'TOTPOT', 'TOTUSJZ', 'ABSNJZH', 'SAVNCPP', 'USFLUX', 'AREA_ACR',
         'MEANPOT', 'R_VALUE', 'SHRGT45', 'MEANGAM', 'MEANJZH', 'MEANGBT', 'MEANGBZ',
         'MEANJZD', 'MEANGBH', 'MEANSHR', 'MEANALP']
sample_data = pd.read_csv('CMEpredict/normalized_testing_12.csv', names=names)
display(sample_data)
```

	Label	Timestamp	NOAA AR NUM	HARP NUM	TOTUSJH	TOTPOT	TOTUSJZ	ABSNJZH	SAVNCPP	USFLUX	...	R_VALUE	SHRGT45	MEANGAM	MEANJZH	M
0	padding	2016-02-10T22:58:09.80Z	12497	6327	-0.0759	0.3866	-0.4445	-0.2113	-0.2948	1.0499	...	-0.1779	0.6987	0.9725	-0.2374	
1	padding	2016-02-10T23:10:09.90Z	12497	6327	-0.0967	0.3649	-0.4693	-0.2790	-0.2643	0.8327	...	-0.1750	0.6904	0.9770	-0.2408	
2	padding	2016-02-10T23:22:09.90Z	12497	6327	-0.0904	0.3496	-0.4161	-0.1743	-0.3262	1.1293	...	-0.1774	0.6946	1.0026	-0.2407	
3	padding	2016-02-10T23:34:09.90Z	12497	6327	-0.1058	0.3376	-0.4049	-0.1645	-0.3412	1.0525	...	-0.1792	0.6842	1.0105	-0.2502	
4	padding	2016-02-10T23:46:09.90Z	12497	6327	-0.1035	0.3401	-0.4028	-0.1468	-0.3443	1.2058	...	-0.1840	0.6868	0.9994	-0.2505	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
716	P	2016-04-17T23:34:41.00Z	12529	6483	0.9327	0.4485	0.4513	0.2630	-0.5814	-0.0546	...	1.2833	0.7819	-0.5989	1.3702	
717	P	2016-04-17T23:46:41.00Z	12529	6483	0.9399	0.4514	0.5107	0.2952	-0.6123	0.1997	...	1.2744	0.7709	-0.5941	1.3693	
718	P	2016-04-17T23:58:41.00Z	12529	6483	0.9489	0.4705	0.5293	0.3388	-0.6220	0.0527	...	1.2723	0.7757	-0.5994	1.3751	
719	P	2016-04-18T00:10:41.00Z	12529	6483	0.9428	0.4584	0.5930	0.5310	-0.6540	0.1240	...	1.2669	0.7690	-0.6131	1.3667	
720	P	2016-04-18T00:22:41.10Z	12529	6483	0.9296	0.4169	0.5439	0.4369	-0.6323	0.1523	...	1.2683	0.7496	-0.6163	1.3471	

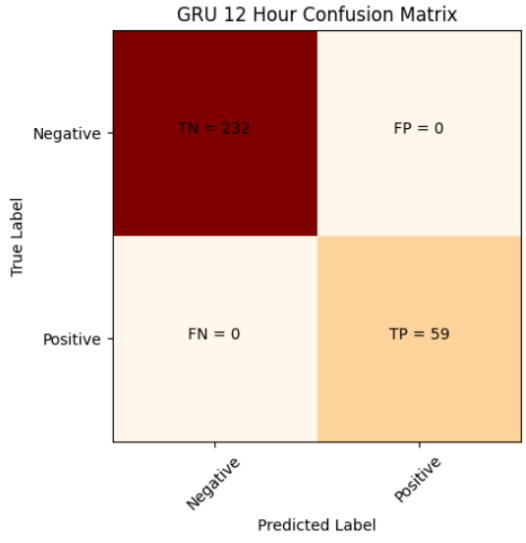
721 rows × 22 columns

- Prediction and Analysis
    - Predicting with Pre-Trained Models

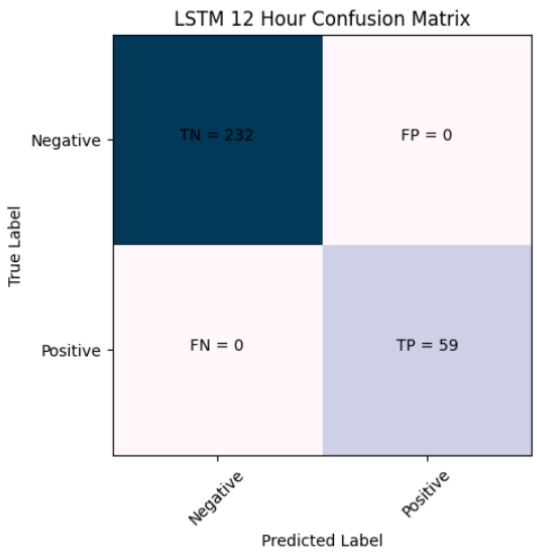
In [4]:

```
import itertools
time_windows = [12, 36, 60] # options are 12, 24, 36, 48, 60
rnn_types = ['gru', 'lstm']
cm_grid = [['TN','FP'], ['FN', 'TP']]

results = {}

for (time_window, type) in itertools.product(time_windows, rnn_types):
    print(f'predicting for {time_window} hour window using {type}')
    model_file = f'CMEpredict/{type}-{time_window}-model.h5'

    n_features, threshold = get_n_features_thresh(type, time_window)

    model = load_model(model_file)

    test_data_file = f'CMEpredict/normalized_testing_{time_window}.csv'
    X_test, y_test, nb_test = load_data(datafile=test_data_file,
                                        series_len=series_len,
                                        start_feature=start_feature,
                                        n_features=n_features,
                                        mask_value=mask_value,
                                        type=type,
                                        time_window=time_window)

    prob = model.predict(X_test,
                         batch_size=batch_size,
                         verbose=False,
                         steps=None)

    bc = [1 if p >= threshold else 0 for p in prob]
    confmat = cm(y_test, bc)
    results[f'{type} {time_window}'] = (y_test, bc)
    plt.imshow(confmat, interpolation='nearest', cmap=(plt.cm.PuBu if type =='lstm' else plt.cm.OrRd))
    classNames = ['Negative','Positive']
    plt.title(f'{type.upper()} {time_window} Hour Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    for i in range(2):
        for j in range(2):
            plt.text(j,i, f'{cm_grid[i][j]} = {confmat[i][j]}', horizontalalignment='center')
    plt.show()
```

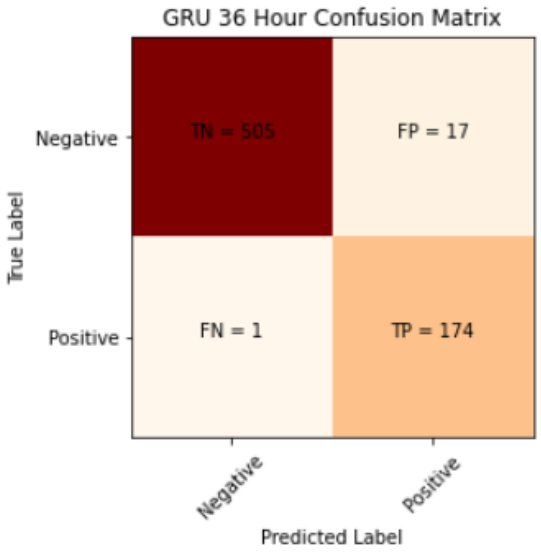
```
predicting for 12 hour window using gru
```



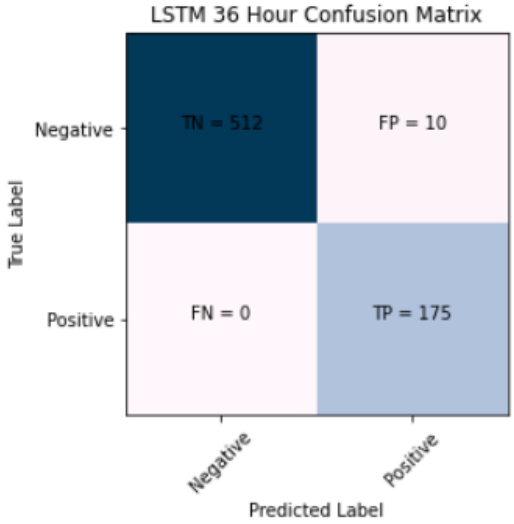

```
predicting for 12 hour window using lstm
```



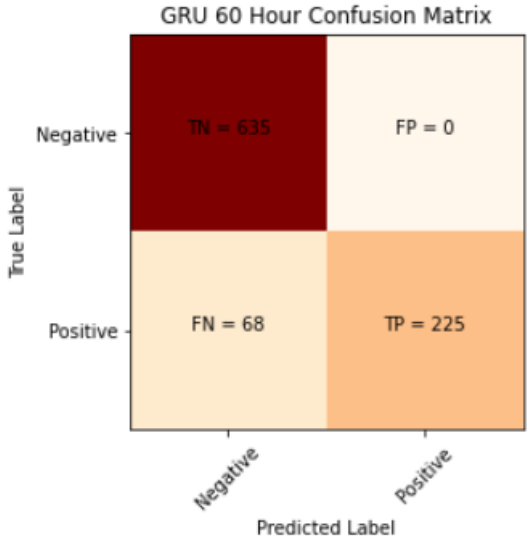
predicting for 36 hour window using gru



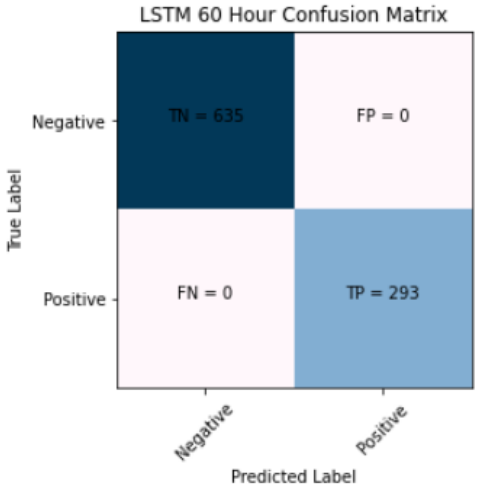
predicting for 36 hour window using lstm



predicting for 60 hour window using gru



predicting for 60 hour window using lstm

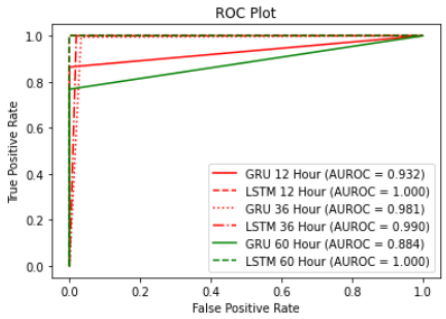


-

- Analysis of Predictions

2.4.2. Analysis of Predictions

In [5]:
```
plt.rc('axes', prop_cycle=(cycler('color', ['r', 'g', 'b', 'y', 'c', 'k']) *
                           cycler('linestyle', ['-', '--', ':', '-.'])))
for name, (y_test, bc) in results.items():
    r_auc = roc_auc_score(y_test, bc)
    r_fpr, r_tpr, _ = roc_curve(y_test, bc)
    plt.plot(r_fpr, r_tpr, label=f'{name.upper()} Hour (AUROC = %0.3f)' % r_auc)
plt.title('ROC Plot')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

plt.legend()

plt.show()
```



- Train your own model

In [6]:
```
# CHANGE THESE VARIABLES#
time_window = 12        # 12, 24, 36, 48, 60
type = 'lstm'           # 'lstm', 'gru'
#########################

train_data_file = f'CMEpredict/normalized_training_{time_window}.csv'
n_features, thresh = get_n_features_thresh(type, time_window)
X_train, y_train, nb_train = load_data(datafile=train_data_file,
                                       series_len=series_len,
                                       start_feature=start_feature,
                                       n_features=n_features,
                                       mask_value=mask_value,
                                       type=type,
                                       time_window=time_window)

class_weights = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
class_weight_ = {0: class_weights[0], 1: class_weights[1]}

if type is 'lstm':
    model = lstm(n_features, series_len)
elif type  is 'gru':
    model = gru(n_features, series_len)

print('training the model, wait until it is finished...')
model.compile(loss='binary_crossentropy',
              optimizer='RMSprop',
              metrics=['accuracy'])

history = model.fit(X_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    verbose=False,
                    shuffle=True,
                    class_weight=class_weight_)
model_file = f'CMEpredict/{type}-{time_window}-model.h5'
model.save(model_file)
print('done training the model')
```

```
training the model, wait until it is finished...
done training the model
```

3.4 Predicting Solar Flares Using a Long Short-term Memory Network

In this notebook, we attempt to use SDO/HMI vector magnetic field data together with flaring history to predict solar flares that would occur in an AR within 24 hr of a given time point, with a deep-learning method, named long short-term memory.

Tool Version-1.0.2: To run this project we can download python version: 3.6.x, and set up the environment either in virtual environment, install the required modules. Launch jupyter notebook and run the cells.

- C Flare Model Training and Testing
    - C Flare Model Training with Default Data

In [1]:
```
print('Loading the train_model function...')
from tensorflow.python.keras import regularizers
from flarepredict_train import train_model
args = {'train_data_file':'data/LSTM_C_sample_run/normalized_training.csv',
        'flare': 'C',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: data/LSTM_C_sample_run/normalized_training.csv
Loading data set...
(84577, 10, 14)
Done loading data...
Training is in progress, please wait until it is done...

Finished training the C flare model, you may use the flarepredict_test.py program to make prediction.
```

- Predicting with Your C Flare Model

In [2]:
```
from flarepredict_test import test_model
args =  {'test_data_file': 'data/LSTM_C_sample_run/normalized_testing.csv',
         'flare': 'C',
         'modelid': 'custom_model_id'}
result_file_name = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/LSTM_C_sample_run/normalized_testing.csv
Loading data set...
(185, 10, 14)
Done loading data...
Formatting and mapping the data...
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

- Reading the Results

```
In [3]:
import pandas as pd
df = pd.read_csv(result_file_name)
df
```

Out[3]:

	Predicted Label	label	flare	timestamp	NOAA	HARP	TOTUSJH	Cdec	TOTUSJZ	Chis1d	USFLUX	TOTBSQ	R_VALUE	TOTPOT	Chis
0	Negative	Negative	B2.8	2017-03-25T01:22:36.70Z	12644	6972	-0.805765	0.000000	-0.509193	0.000000	-0.925044	-0.599012	-1.136922	-0.855732	0.000000
1	Negative	Negative	B2.8	2017-03-25T02:22:36.70Z	12644	6972	-0.769690	0.000000	-0.473642	0.000000	-0.900514	-0.592492	-1.180087	-0.822543	0.000000
2	Negative	Negative	B2.8	2017-03-25T04:22:36.80Z	12644	6972	-0.755696	0.000000	-0.441264	0.000000	-0.900614	-0.590719	-1.414541	-0.819419	0.000000
3	Negative	Negative	B2.8	2017-03-25T15:22:36.60Z	12644	6972	-0.713604	0.000000	-0.505940	0.000000	-0.871843	-0.568816	-0.503298	-0.755192	0.000000
4	Negative	Negative	B2.8	2017-03-25T16:22:36.60Z	12644	6972	-0.681474	0.000000	-0.348887	0.000000	-0.853421	-0.556959	-0.601962	-0.720032	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
180	Positive	Positive	M5.3	2017-04-02T07:22:37.90Z	12644	6972	1.735370	0.126305	1.325002	0.111111	0.833936	1.003125	0.600823	1.487252	0.045977
181	Positive	Positive	M2.3	2017-04-02T08:22:37.90Z	12644	6972	1.810731	0.116207	1.202598	0.111111	0.854023	1.118890	0.716461	1.526074	0.045977
182	Positive	Positive	M2.3	2017-04-02T09:22:37.90Z	12644	6972	1.799658	0.106915	1.588251	0.111111	0.897038	1.193614	0.740491	1.437500	0.045977
183	Positive	Positive	M2.3	2017-04-02T10:22:37.90Z	12644	6972	1.819823	0.098367	1.659994	0.111111	0.943544	1.256551	0.717097	1.448933	0.045977
184	Positive	Positive	M2.3	2017-04-02T11:22:37.90Z	12644	6972	1.968782	0.090502	1.791367	0.111111	1.001174	1.383610	0.689125	1.500635	0.045977

185 rows × 20 columns

- M Flare Model Training and Testing
  - M Flare Model Training with Default Data

```
In [4]:
print('Loading the train_model function...')
from flarepredict_train import train_model
args = {'train_data_file':'data/LSTM_M_sample_run/normalized_training.csv',
        'flare': 'M',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: data/LSTM_M_sample_run/normalized_training.csv
Loading data set...
(84742, 10, 22)
Done loading data...
Training is in progress, please wait until it is done...

Finished training the M flare model, you may use the flarepredict_test.py program to make prediction.
```

  - Predicting with Your M Flare Model

```
In [5]:
from flarepredict_test import test_model
args =  {'test_data_file': 'data/LSTM_M_sample_run/normalized_testing.csv',
         'flare': 'M',
         'modelid': 'custom_model_id'}
result_file_name = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/LSTM_M_sample_run/normalized_testing.csv
Loading data set...
(212, 10, 22)
Done loading data...
Formatting and mapping the data...
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

- Reading the Results

In [6]:
```
import pandas as pd
df = pd.read_csv(result_file_name)
df
```

Out[6]:

	Predicted Label	Label	flare	timestamp	NOAA	HARP	TOTUSJH	TOTUSJZ	TOTPOT	TOTBSQ	...	Xmax1d	Mhis	R_VALUE	Mdec	MEANPC
0	Negative	Negative	C1.0	2015-04-04T05:34:17.60Z	12320	5415	-0.104048	-0.301538	-0.137916	-0.364581	...	0.000000	0.000000	-0.841946	0.000000	0.6874
1	Negative	Negative	C1.0	2015-04-04T06:34:17.60Z	12320	5415	-0.078954	-0.175091	-0.094344	-0.356320	...	0.000000	0.000000	-0.792015	0.000000	0.4321
2	Negative	Negative	C1.0	2015-04-04T07:34:17.60Z	12320	5415	-0.043782	-0.338201	-0.018729	-0.349030	...	0.000000	0.000000	-0.798981	0.000000	0.3286
3	Negative	Negative	C1.0	2015-04-04T08:34:17.60Z	12320	5415	0.018644	-0.306688	0.096848	-0.326073	...	0.000000	0.000000	-0.690922	0.000000	0.6624
4	Negative	Negative	C1.0	2015-04-04T09:34:17.60Z	12320	5415	0.055247	-0.372511	0.116056	-0.300240	...	0.000000	0.000000	-0.600492	0.000000	0.7322
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
207	Negative	Negative	C2.9	2015-04-13T15:34:18.70Z	12320	5415	0.947701	-0.229454	0.790858	0.109802	...	0.006882	0.038462	-0.572136	0.000007	1.0567
208	Negative	Negative	C2.9	2015-04-13T16:34:18.70Z	12320	5415	0.945490	-0.256896	0.851387	0.127448	...	0.006882	0.038462	-0.496255	0.000007	1.0887
209	Negative	Negative	C2.9	2015-04-13T17:34:18.70Z	12320	5415	0.826441	-0.314707	0.766035	0.110474	...	0.006882	0.038462	-0.503343	0.000006	1.0881
210	Negative	Negative	C2.9	2015-04-13T18:34:18.70Z	12320	5415	0.944913	-0.175675	0.811451	0.161015	...	0.006882	0.038462	-0.487871	0.000006	1.0520
211	Negative	Negative	C2.9	2015-04-13T19:34:18.80Z	12320	5415	0.959147	-0.067101	0.905028	0.203846	...	0.006882	0.038462	-0.399229	0.000005	1.0317

212 rows × 28 columns

- M5 Flare Model Training and Testing
  - M5 Flare Model Training with Default Data

In [7]:
```
print('Loading the train_model function...')
from flarepredict_train import train_model
args = {'train_data_file':'data/LSTM_M5_sample_run/normalized_training.csv',
        'flare': 'M5',
       'modelid': 'custom_model_id'
      }
train_model(args)
```

```
Loading the train_model function...
Starting training with a model with id: custom_model_id training data file: data/LSTM_M5_sample_run/normalized_training.csv
Loading data set...
(84798, 10, 20)
Done loading data...
Training is in progress, please wait until it is done...

Finished training the M5 flare model, you may use the flarepredict_test.py program to make prediction.
```

  - Predicting with Your M5 Flare Model

In [8]:
```
from flarepredict_test import test_model
args =  {'test_data_file': 'data/LSTM_M5_sample_run/normalized_testing.csv',
         'flare': 'M5',
         'modelid': 'custom_model_id'}
result_file_name = test_model(args)
```

```
Starting testing with a model with id: custom_model_id testing data file: data/LSTM_M5_sample_run/normalized_testing.csv
Loading data set...
(189, 10, 20)
Done loading data...
Formatting and mapping the data...
Prediction is in progress, please wait until it is done...
Finished the prediction task..
```

- Reading the Results

In [9]:
```
import pandas as pd
df = pd.read_csv(result_file_name)
df
```

Out[9]:

	Predicted Label	label	flare	timestamp	NOAA	HARP	TOTUSJH	SAVNCPP	ABSNJZH	TOTPOT	...	Edec	Mhis	Xmax1d	Mdec	AREA_A
0	Positive	Negative	N	2017-08-30T08:58:42.90Z	12673	7115	-0.498770	-0.503428	-0.374758	-0.578207	...	0.000000	0.000000	0.000000	0.000000	2.485
1	Positive	Negative	N	2017-08-30T09:58:42.90Z	12673	7115	-0.503880	-0.510741	-0.418607	-0.588074	...	0.000000	0.000000	0.000000	0.000000	2.535
2	Positive	Negative	N	2017-08-30T10:58:42.90Z	12673	7115	-0.488118	-0.504173	-0.449640	-0.571011	...	0.000000	0.000000	0.000000	0.000000	2.289
3	Positive	Negative	N	2017-08-30T11:58:42.80Z	12673	7115	-0.481332	-0.500172	-0.444952	-0.574907	...	0.000000	0.000000	0.000000	0.000000	2.316
4	Positive	Negative	N	2017-08-30T12:58:42.80Z	12673	7115	-0.473880	-0.501714	-0.419139	-0.567325	...	0.000000	0.000000	0.000000	0.000000	2.338
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
184	Negative	Negative	M1.3	2017-09-08T01:58:41.80Z	12673	7115	4.957265	3.600708	5.360394	3.316161	...	0.173517	0.692308	0.139785	0.306003	2.061
185	Negative	Negative	M1.2	2017-09-08T02:58:41.80Z	12673	7115	4.998832	3.638885	5.449238	3.350028	...	0.172529	0.730769	0.139785	0.449894	2.045
186	Negative	Positive	M8.1	2017-09-08T03:58:41.80Z	12673	7115	4.875287	3.552344	5.284544	3.316022	...	0.171174	0.769231	0.139785	0.586064	2.201
187	Negative	Positive	M8.1	2017-09-08T04:58:41.80Z	12673	7115	4.843132	3.583137	5.129436	3.522198	...	0.157488	0.769231	0.139785	0.539205	2.217
188	Negative	Positive	M8.1	2017-09-08T05:58:41.80Z	12673	7115	4.762235	3.627277	5.090359	3.258578	...	0.152696	0.769231	0.139785	0.496092	1.978

189 rows × 26 columns

3.5 Predicting Solar Energetic Particles Using SDO/HMI Vector Magnetic Data Products and a Bidirectional LSTM Network

Tool Version-1.4: This tool leverages python deep learning to describe the steps on how to use the BiLSTM tool to predict the binary SEP classification problems we are sovling.To run this project we can download python version: 3.9.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

- Installation

In [1]:
```
#supress warning messages
import warnings
warnings.filterwarnings('ignore')
print('Importing packages..')
# Data manipulation
import pandas as pd
import numpy as np
import os
print('Packages imported')
#make sure the scripts are executed in the correct pacakage location.
if os.path.exists('SEP_Package'):
    print('Changing working directory to SEP_Package..')
    os.chdir('SEP_Package')
import sys
sys.path.append('.')

```

```
Importing packages..
Packages imported
Changing working directory to SEP_Package..
```

- Predicting with Pretrained Models

In [2]:
```
#Test default models for FC_S for 12-72 hours.
from SEP_test import test

models_directory='default_models'
print('Test default models for first classification type FC_S and for all time windows: 12 to 72 hours.')
classification_type='FC_S'
starting_time_window=12
ending_time_window= 72
test(classification_type,starting_time_window,ending_time_window+1,models_directory=models_directory)
```

```
WARNING: GPU device not found.
WARNING: GPU device not found.
Test default models for first classification type FC_S and for all time windows: 12 to 72 hours.
Running classification test type: FC_S  training for h = 12 hour ahead
testing data file: data/events_fc_testing_12.csv
Loading the model and its weights.
Loading weights from model dir: default_models\sep_model_fc_12hr
Building model for: default_models\sep_model_fc_12hr\model_weights
Loading weights from: default_models\sep_model_fc_12hr\model_weights
Loading data from data file: data/events_fc_testing_12.csv
Prediction and calibration..
Saving result to file: results\SEP_prediction_results_FC_S_12.csv
Saving the performance metrics to files: default_results\SEP_performance_metrics_BiLSTM_FC_S_12.csv
------------------------------------------------------------

Running classification test type: FC_S  training for h = 24 hour ahead
testing data file: data/events_fc_testing_24.csv
Loading the model and its weights.
Loading weights from model dir: default_models\sep_model_fc_24hr
```

In [3]:

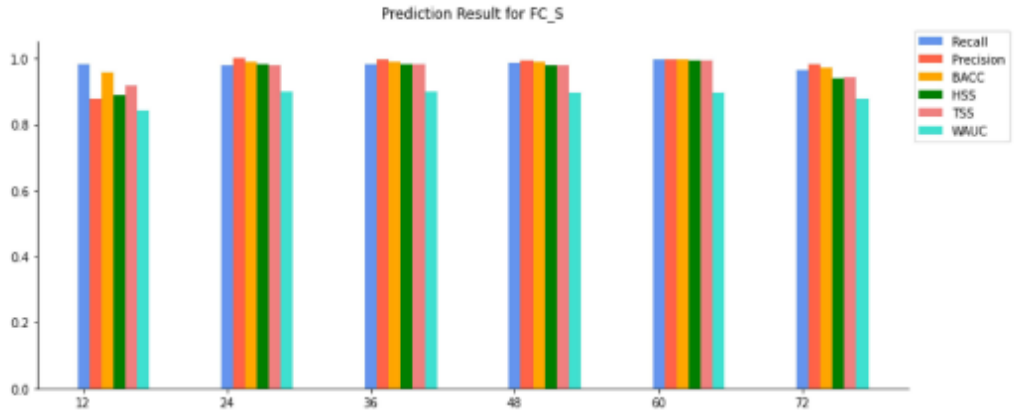
```
#Testing default models for F_S for 12-72 hours.
from SEP_test import test

models_directory='default_models'
print('Test default models for second classification type F_S and for all time windows: 12 to 72 hours.')
classification_type='F_S'
test(classification_type,starting_time_window,ending_time_window+1,models_directory=models_directory)
```

```
Test default models for second classification type F_S and for all time windows: 12 to 72 hours.
Running classification test type: F_S  training for h = 12 hour ahead
testing data file: data/events_f_testing_12.csv
Loading the model and its weights.
Loading weights from model dir: default_models\sep_model_f_12hr
Building model for: default_models\sep_model_f_12hr\model_weights
Loading weights from: default_models\sep_model_f_12hr\model_weights
Loading data from data file: data/events_f_testing_12.csv
Prediction and calibration..
Saving result to file: results\SEP_prediction_results_F_S_12.csv
Saving the performance metrics to files: default_results\SEP_performance_metrics_BiLSTM_F_S_12.csv
------------------------------------------------------------

Running classification test type: F_S  training for h = 24 hour ahead
testing data file: data/events_f_testing_24.csv
Loading the model and its weights.
Loading weights from model dir: default_models\sep_model_f_24hr
Building model for: default_models\sep_model_f_24hr\model_weights
Loading weights from: default_models\sep_model_f_24hr\model_weights
```

- Plotting the Pretrained Models Results

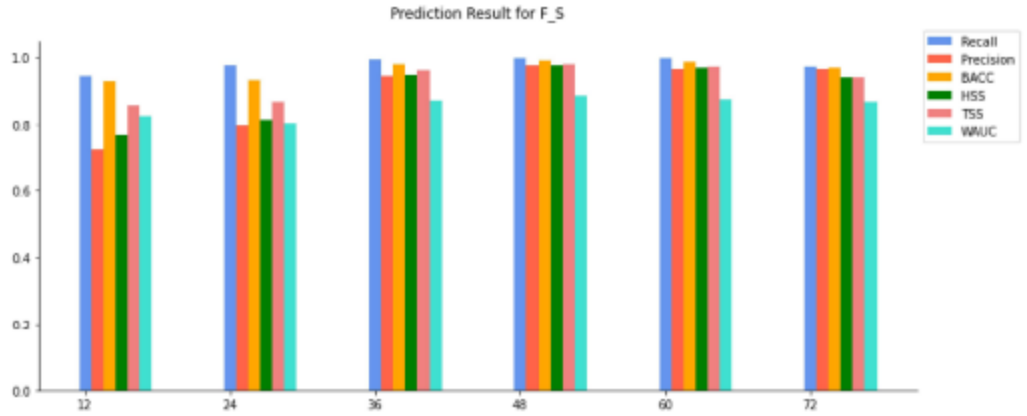
In [4]:

```
from SEP_utils import plot_result_metrics
#Plotting result for FC_S
plot_result_metrics('FC_S',result_dir='default_results')
```



In [5]:

```
#Plotting results metrics for F_S
from SEP_utils import plot_result_metrics
plot_result_metrics('F_S',result_dir='default_results')
```



- BiLSTM Model Training and Testing Example

In [12]:

```
#Training for FC_S 12-72 hours on sample data.
print('Loading the train_model function...')
from SEP_train import train_model
print('Train model for FC_S classification and time window h=12-72')
classification_type='FC_S'
starting_time_window=12
ending_time_window= 72

train_model(classification_type,starting_time_window,ending_time_window+1)
```

```
Loading the train_model function...
Train model for FC_S classification and time window h=12-72


Running classification type: FC_S  training for h = 12 hour ahead
Loading data from data file: data/events_fc_training_12.csv
Loading data from data file: data/events_fc_testing_12.csv
Epoch 1/5
1/1 [==============================] - 2s 2s/step
Epoch 2/5
1/1 [==============================] - 2s 2s/step

Epoch 3/5
1/1 [==============================] - 3s 3s/step

Epoch 4/5
1/1 [==============================] - 3s 3s/step
Epoch 5/5
1/1 [==============================] - 3s 3s/step
```

In [14]:

```
#Training for FS_S 12-72 hours on sample data.
print('Loading the train_model function...')
from SEP_train import train_model
print('Train model for F_S classification and time window h=12-72')
classification_type='F_S'
starting_time_window=12
ending_time_window= 72

train_model(classification_type,starting_time_window,ending_time_window+1)
```

```
Loading the train_model function...
Train model for F_S classification and time window h=12-72


Running classification type: F_S  training for h = 12 hour ahead
Loading data from data file: data/events_f_training_12.csv
Loading data from data file: data/events_f_testing_12.csv
Epoch 1/5
1/1 [==============================] - 4s 4s/step
Epoch 2/5
1/1 [==============================] - 4s 4s/step
Epoch 3/5
1/1 [==============================] - 5s 5s/step
Epoch 4/5
1/1 [==============================] - 6s 6s/step

Epoch 5/5
1/1 [==============================] - 6s 6s/step
```

- Predicting with Your Trained BiLSTM Model

In [15]:

```
#Test trained model for FC_S and 12-hour
from SEP_test import test

models_directory='models'
print('Test the trained models for first classification type FC_S and for time windows h=12.')
classification_type='FC_S'
starting_time_window=12
ending_time_window= 72
test(classification_type,starting_time_window,ending_time_window+1,models_directory=models_directory)
```

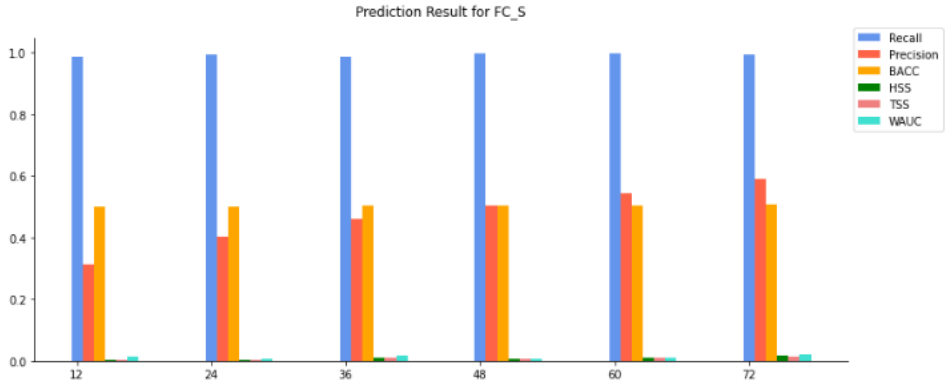
```
Test the trained models for first classification type FC_S and for time windows h=12.
Running classification test type: FC_S  training for h = 12 hour ahead
testing data file: data/events_fc_testing_12.csv
Loading the model and its weights.
Loading weights from model dir: models\sep_model_fc_12hr
Building model for: models\sep_model_fc_12hr\model_weights
Loading weights from: models\sep_model_fc_12hr\model_weights
Loading data from data file: data/events_fc_testing_12.csv
Prediction and calibration..
Saving result to file: results\SEP_prediction_results_FC_S_12.csv
Saving the performance metrics to files: results\SEP_performance_metrics_BiLSTM_FC_S_12.csv
------------------------------------------------------------

Running classification test type: FC_S  training for h = 24 hour ahead
testing data file: data/events_fc_testing_24.csv
Loading the model and its weights.
Loading weights from model dir: models\sep_model_fc_24hr
Building model for: models\sep_model_fc_24hr\model_weights
Loading weights from: models\sep_model_fc_24hr\model_weights
```

In [16]:
```
#Test trained model for F_S and 12-hour
from SEP_test import test

models_directory='models'
print('Test the trained models for second classification type F_S and for time windows h=12.')
classification_type='F_S'
test(classification_type,starting_time_window,ending_time_window+1,models_directory=models_directory)
```

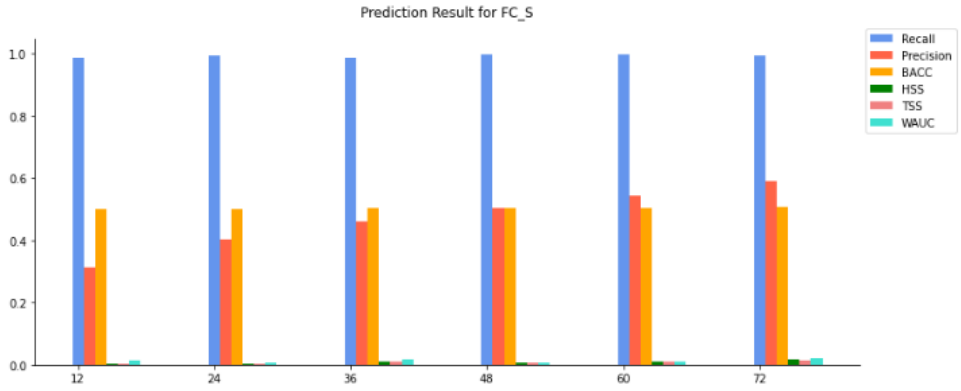
```
Test the trained models for second classification type F_S and for time windows h=12.
Running classification test type: F_S  training for h = 12 hour ahead
testing data file: data/events_f_testing_12.csv
Loading the model and its weights.
Loading weights from model dir: models\sep_model_f_12hr
Building model for: models\sep_model_f_12hr\model_weights
Loading weights from: models\sep_model_f_12hr\model_weights
Loading data from data file: data/events_f_testing_12.csv
Prediction and calibration..
Saving result to file: results\SEP_prediction_results_F_S_12.csv
Saving the performance metrics to files: results\SEP_performance_metrics_BiLSTM_F_S_12.csv
-----------------------------------------------------------

Running classification test type: F_S  training for h = 24 hour ahead
testing data file: data/events_f_testing_24.csv
Loading the model and its weights.
Loading weights from model dir: models\sep_model_f_24hr
Building model for: models\sep_model_f_24hr\model_weights
Loading weights from: models\sep_model_f_24hr\model_weights
```

- Plotting the Results for Your Trained Model

In [17]:
```
#Plotting results for trained model for FC_S and time window =12-72
from SEP_utils import plot_result_metrics
plot_result_metrics('FC_S')
```



In [18]:
```
#Plotting results for trained model for F_S and time window =12-72
from SEP_utils import plot_result_metrics
plot_result_metrics('FC_S')
```



3.6 Forecasting the Disturbance Storm Time Index with Bayesian Deep Learning

Tool Version-1.1: This tool leverages python deep learning to describe the steps on how to use the DSTT tool to forecast the Dst index for 1 to 6 hours ahead.To run this project we can download python version: 3.9.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

- Library import

In [1]:
```
#supress warning messages
import warnings
warnings.filterwarnings('ignore')
print('Importing packages..')
# Data manipulation
import pandas as pd
import numpy as np
import os
print('Packages imported')
#make sure the scripts are executed in the correct pacakage location.
if os.path.exists('DSTT_Package'):
    print('Changing working directory to DSTT_Package..')
    os.chdir('DSTT_Package')
import sys
sys.path.append('.')

```

```
Importing packages..
Packages imported
Changing working directory to DSTT_Package..
```

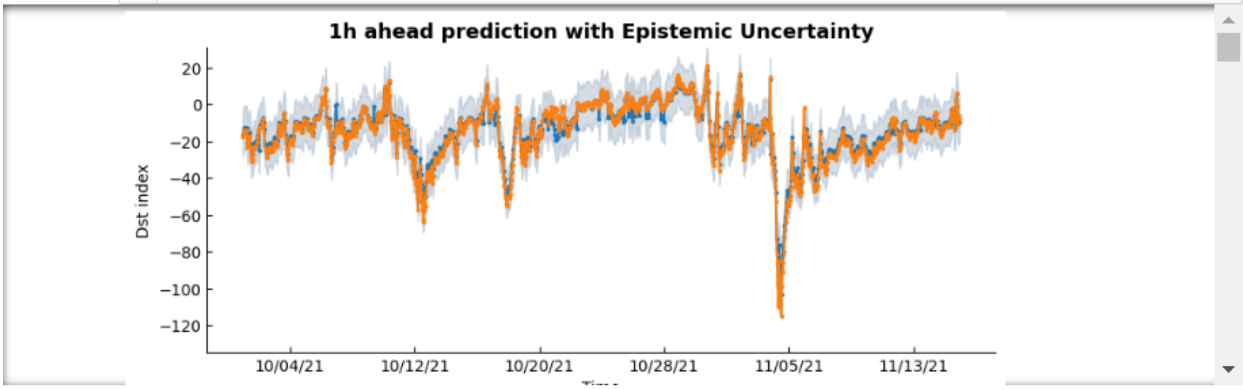
- Predicting with Pretrained Models
    - Testing with Pretrained models

In [2]:
```
#Test default models for 1-6 hours.
from DSTT_test import test

models_directory='default_models'
print('Test default models for Dst forecasting for 1-6 hours ahead.')
start_hour=1
end_hour=6
test(start_hour,end_hour+1,models_directory=models_directory)

```

```
WARNING: GPU device not found.
Test default models for Dst forecasting for 1-6 hours ahead.
Running testing for h = 1 hour ahead
1/1 [==============================] - 0s 381ms/step

Uncertainty Quantification
  1/100  [===== Uncertainty Quantification ======]  -    1/100 %
  2/100  [===== Uncertainty Quantification ======]  -    2/100 %
  3/100  [===== Uncertainty Quantification ======]  -    3/100 %
  4/100  [===== Uncertainty Quantification ======]  -    4/100 %
  5/100  [===== Uncertainty Quantification ======]  -    5/100 %
  6/100  [===== Uncertainty Quantification ======]  -    6/100 %
  7/100  [===== Uncertainty Quantification ======]  -    7/100 %
  8/100  [===== Uncertainty Quantification ======]  -    8/100 %
  9/100  [===== Uncertainty Quantification ======]  -    9/100 %
 10/100  [===== Uncertainty Quantification ======]  -   10/100 %
 11/100  [===== Uncertainty Quantification ======]  -   11/100 %
 12/100  [===== Uncertainty Quantification ======]  -   12/100 %
```

- Plotting the Pretrained Models Results

In [3]:
```
from DSTT_plot_results_figures import plot_figures

figures_dir='default_figures'
results_dir='default_results'
print('Plotting figures for default models results for Dst forecasting for 1-6 hours ahead.')
start_hour=1
end_hour=6

plot_figures(start_hour, end_hour+1,show_figures=True,figures_dir = figures_dir,results_dir=results_dir)

```



- DSTT Model Training and Testing Example
  - DSTT Model Training with Sample Data

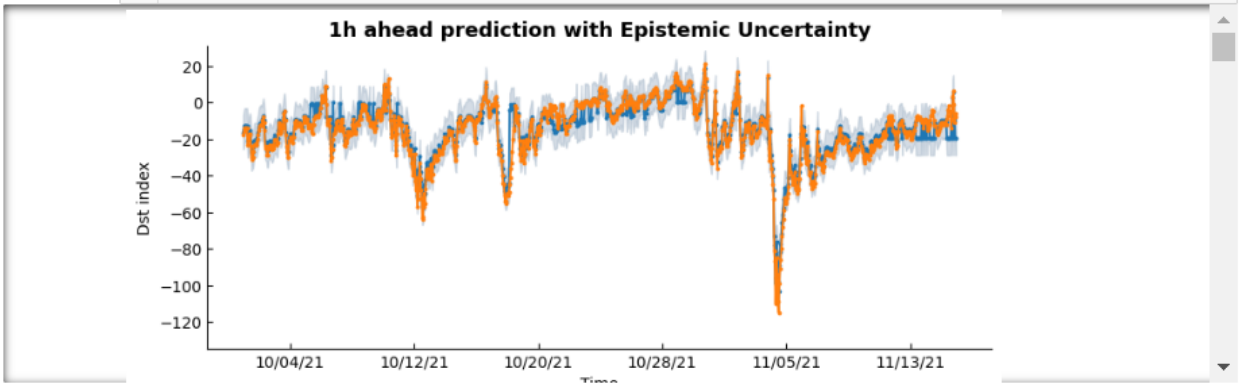
In [4]:
```
#Training for FC_S 12-72 hours on sample data.
print('Loading the train_model function...')
from DSTT_train import train_model
print('Train custom model for h=1-6')
start_hour=1
end_hour=6
#set the number of epochs=100
epochs=100
train_model(start_hour,end_hour+1,epochs=epochs)

```

  - Predicting with Your Trained DSTT Model

In [5]:
```
#Test default models for 1-6 hours.
from DSTT_test import test

models_directory='models'
print('Test your trained models for Dst forecasting for 1-6 hours ahead.')
start_hour=1
end_hour=6
test(start_hour,end_hour+1,models_directory=models_directory)

```

- Plotting the Results for Your Trained Model

In [6]:
```
from DSTT_plot_results_figures import plot_figures

figures_dir='figures'
results_dir='results'
print('Plotting figures for default models results for Dst forecasting for 1-6 hours ahead.')
start_hour=1
end_hour=6

plot_figures(start_hour, end_hour+1,show_figures=True,figures_dir = figures_dir,results_dir=results_dir)

```



3.7 Reconstruction of Total Solar Irradiance by Deep Learning

Tool Version-1.6: To run this project we can download python version: 3.8.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

Initially to set the file path we run:

```
(entsi) C:\Users\Kavya\OneDrive\Documents\Full Stack\Project\TSInet-irradiance-reconstruction-v1.6\ccsc-tools-TSInet-irradiance-reconstruction-3efd99f>python tsinet_reconstructor.py -n TCTE
Python version: 3.8.10
Tensorflow backend version: 2.4.0
model_dir: models

Please wait while reconstructing...
Loading model file: models\tsinet_model
Reconstruction will be performed using the model file: models\tsinet_model
Processing with k steps = 5
Finished the reconstruction process.
Results are written to: results\TCTE_result.csv
```

- Library Import

In [1]:
```
#supress warning messages
import warnings
warnings.filterwarnings('ignore')
print('Importing packages..')
# Data manipulation
import pandas as pd
import numpy as np
import os
print('Packages imported')
```

```
Importing packages..
Packages imported
```

- TSInet Workflow and Results
    - Reconstructing with Pretrained Models

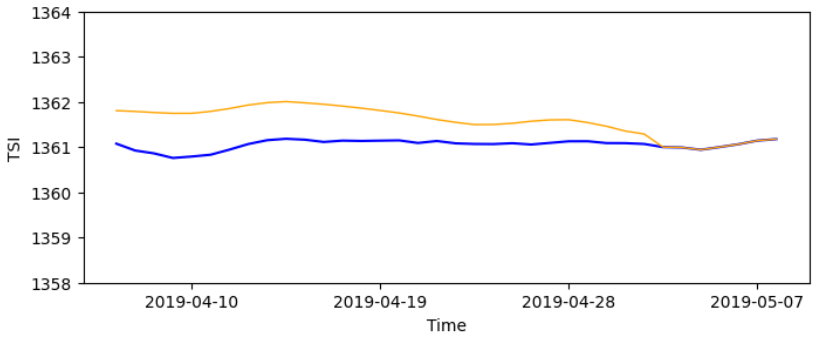
In [2]:
```
#Test default models.
from tsinet_reconstructor import *
dataset_name='TCTE'
model_directory='default_model'
print('Testing the default model with data set:', dataset_name)
args={}

args['dataset_name'] = dataset_name
args['model_dir']  = model_directory
print('Process the require arguments..')
process_args(args)

reconstruct_tsinet()
```

```
Python version: 3.8.10
Tensorflow backend version: 2.4.0
Testing the default model with data set: TCTE
Process the require arguments..
model_dir: default_model

Please wait while reconstructing...
Loading model file: default_model\tsinet_model
Reconstruction will be performed using the model file: default_model\tsinet_model
Processing with k steps = 5
Finished the reconstruction process.
Results are written to: results\TCTE_result.csv
```

- Plotting the Pretrained Models Results

In [3]:
```
from tsinet_utils import plot_figures

dataset_name='TCTE'
result_file_name = 'results/tsinet_result_' + dataset_name + '.csv'
print('Plotting figures for default models results for TSI reconstruction of data set:', dataset_name)
plot_figures(result_file_name)

```

```
Plotting figures for default models results for TSI reconstruction of data set: TCTE

*NOTE: the figure scale is large compared to the difference between the observed and predicted values, therefore you may see a
big difference, but in actuality, the absolute difference between the values are very small.
```



- TSInet Model Training and Testing Example
  - TSInet Model Training with Sample Data

In [4]:
```
#Training for  sample data.
from tsinet_train import *
print('Training custom model')
training_file='train_data/SORCE_TSI.csv'
epochs=10
args={}
args['training_file'] = training_file
args['epochs'] = epochs

print('Process the require arguments..')
process_args(args)

train_tsinet()

```

```
Training custom model
Process the require arguments..
Epoch 1/10
356/356 [==============================] - 2s 3ms/step - loss: 0.0426
Epoch 2/10
356/356 [==============================] - 1s 3ms/step - loss: 0.0011
Epoch 3/10
356/356 [==============================] - 1s 3ms/step - loss: 0.0011
Epoch 4/10
356/356 [==============================] - 1s 3ms/step - loss: 6.9023e-04
Epoch 5/10
356/356 [==============================] - 1s 3ms/step - loss: 4.1562e-04
Epoch 6/10
356/356 [==============================] - 1s 3ms/step - loss: 3.4664e-04
Epoch 7/10
356/356 [==============================] - 1s 3ms/step - loss: 4.7520e-04
Epoch 8/10
356/356 [==============================] - 1s 3ms/step - loss: 4.5817e-04
Epoch 9/10
356/356 [==============================] - 1s 3ms/step - loss: 4.5966e-04
Epoch 10/10
356/356 [==============================] - 1s 3ms/step - loss: 4.2797e-04
Saving the TSInet model
idky models\tsinet_model.h5
The model is saved in the models directory
Finished TSInet training...
```

- Reconstructing with Your Trained TSInet Model

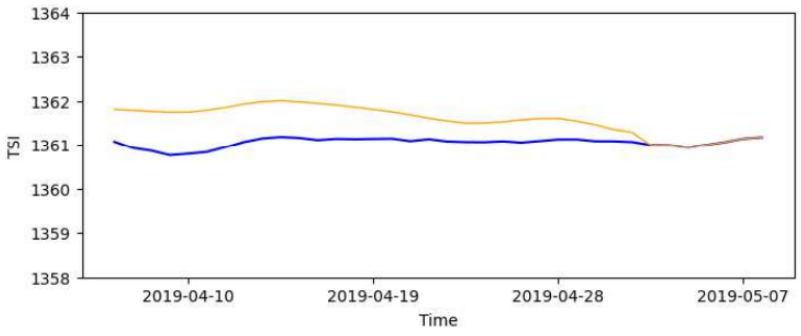
In [5]:
```
#Test default models.
from tsinet_reconstructor import *
import os
dataset_name='TCTE'
model_directory='models'
result_file_name = 'results' +os.sep +'sinet_result_'+dataset_name+'.csv'
print('Testing the custom model with data set:', dataset_name)
args={}

args['dataset_name'] = dataset_name
args['model_dir']  = model_directory
args['result_file_name'] = result_file_name

print('Process the require arguments..')
process_args(args)

reconstruct_tsinet()
```

```
Testing the custom model with data set: TCTE
Process the require arguments..
model_dir: models

Please wait while reconstructing...
Loading model file: models\tsinet_model
Reconstruction will be performed using the model file: models\tsinet_model
Processing with k steps = 5
Finished the reconstruction process.
Results are written to: results\sinet_result_TCTE.csv
```

- Plotting the Results for Your Trained Model

In [6]:
```
from tsinet_utils import plot_figures

dataset_name='TCTE'
result_file_name = 'results/tsinet_result_' + dataset_name + '.csv'
print('Plotting figures for default models results for TSI reconstruction of data set:', dataset_name)
plot_figures(result_file_name)
```

```
Plotting figures for default models results for TSI reconstruction of data set: TCTE

*NOTE: the figure scale is large compared to the difference between the observed and predicted values, therefore you may see a
big difference, but in actuality, the absolute difference between the values are very small.
```



3.8 A Transformer-Based Framework for Geomagnetic Activity Prediction

Tool Version-1.1: This tool forecasts the occurrance of the planetary Kp index for the next 1, 2, 3, 4, 5, 6, 7, 8, or 9 hours ahead. The tools also provide data and model uncertainty quantification. To run this project we can download python version: 3.9.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

- Library Import

In [1]:
```
#supress warning messages
import warnings
warnings.filterwarnings('ignore')
print('Importing packages..')
# Data manipulation
import pandas as pd
import numpy as np
import os
print('Packages imported')
#make sure the scripts are executed in the correct pacakage location.
import sys
sys.path.append('.')

```

```
Importing packages..
Packages imported
```

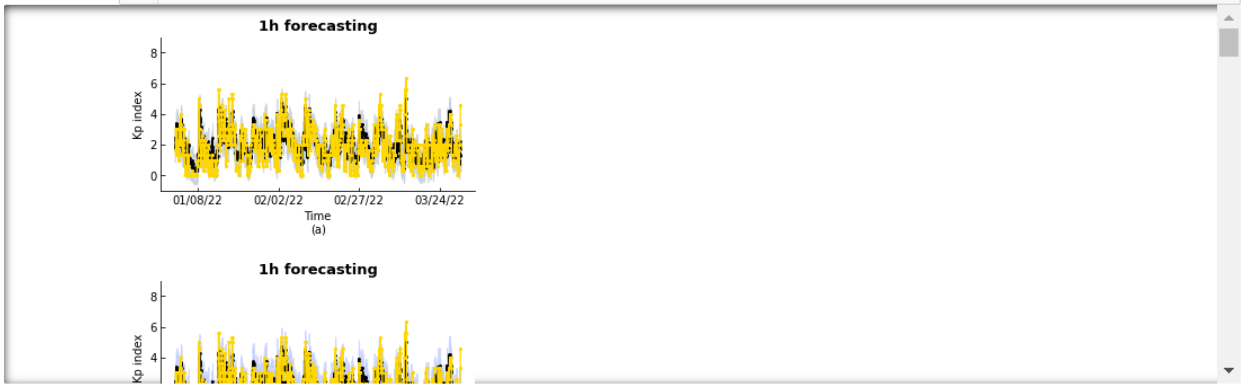
- KpNet Workflow and Results
    - Predicting with Pretrained Models

In [2]:
```
#Test default models for 4hour ahead.
from KpNet_test import test

models_directory='default_models'
print('Test default models for Kp forecasting for 1-9 hours ahead.')
start_hour=4
end_hour=4
test(start_hour,end_hour+1,models_directory=models_directory)

```

```
Test default models for Kp forecasting for 1-9 hours ahead.
Running testing for h = 4 hour ahead
1/1 [==============================] - 11s 11s/step

Uncertainty Quantification
  1/100  [===== Uncertainty Quantification ======]  -    1/100 %
  2/100  [===== Uncertainty Quantification ======]  -    2/100 %
  3/100  [===== Uncertainty Quantification ======]  -    3/100 %
  4/100  [===== Uncertainty Quantification ======]  -    4/100 %
  5/100  [===== Uncertainty Quantification ======]  -    5/100 %
  6/100  [===== Uncertainty Quantification ======]  -    6/100 %
  7/100  [===== Uncertainty Quantification ======]  -    7/100 %
  8/100  [===== Uncertainty Quantification ======]  -    8/100 %
  9/100  [===== Uncertainty Quantification ======]  -    9/100 %
 10/100  [===== Uncertainty Quantification ======]  -   10/100 %
 11/100  [===== Uncertainty Quantification ======]  -   11/100 %
 12/100  [===== Uncertainty Quantification ======]  -   12/100 %
 13/100  [===== Uncertainty Quantification ======]  -   13/100 %
```

- Plotting the Pretrained Models Results

In [3]:
```
from KpNet_plot_results_figures import plot_figures

figures_dir='default_figures'
results_dir='default_results'
print('Plotting figures for default models results for Kp forecasting for 1-9 hours ahead.')
start_hour=1
end_hour=9

plot_figures(start_hour, end_hour+1,show_figures=True,figures_dir = figures_dir,results_dir=results_dir)

```

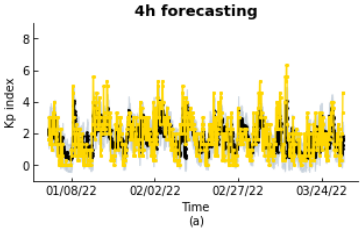


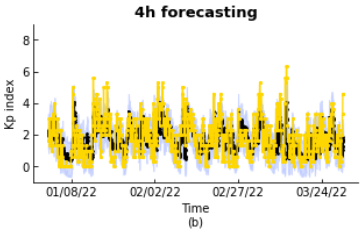
- KpNet Model Training and Testing Example
  - KpNet Model Training with Sample Data

In [4]:
```
#Training for F 4 hour ahead on sample data.
print('Loading the train_model function...')
from KpNet_train import train_model
print('Train custom model for h=4')
start_hour=4
end_hour=4
#set the number of epochs=10
epochs=100
train_model(start_hour,end_hour+1,epochs=epochs)

```

- Predicting with Your Trained KpNet Model

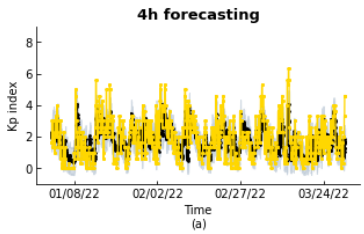
In [5]:
```
#Test custom model for 4-hour ahread.
from KpNet_test import test

models_directory='models'
print('Test your trained models for Kp forecasting for 6-hour ahead.')
start_hour=4
end_hour=4
test(start_hour,end_hour+1,models_directory=models_directory)

```

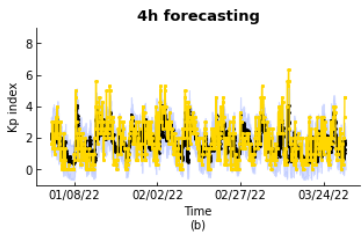




- Plotting the Results for Your Trained Model

In [6]:
```
from KpNet_plot_results_figures import plot_figures

figures_dir='figures'
results_dir='results'
print('Plotting figures for default models results for KP forecasting for 4 hours ahead.')
start_hour=4
end_hour=4

plot_figures(start_hour, end_hour+1,show_figures=True,figures_dir = figures_dir,results_dir=results_dir)


```





3.9 Predicting CME Arrival Time through Data Integration and Ensemble Learning

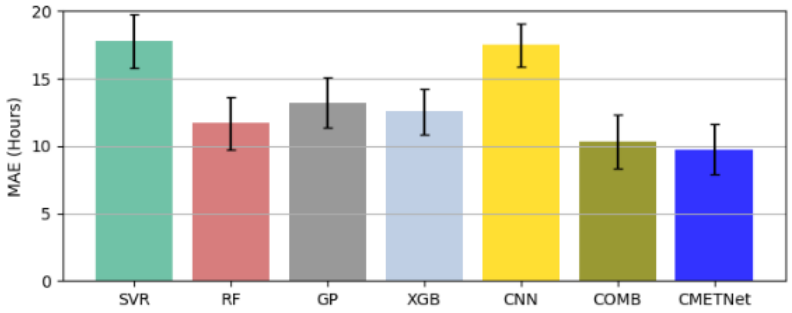
Tool Version-1.1: In this notebook we provide an overview of the CMETNet system to demonstrate how to predict the arrival time of CMEs using machine learning (ML) and data include CME features, solar wind parameters and CME images obtained from the SOHO/LASCO C2 coronagraph. To run this project we can download python version: 3.8.x, and set up the environment either in virtual environment or in miniconda. Launch jupyter notebook and run the cells.

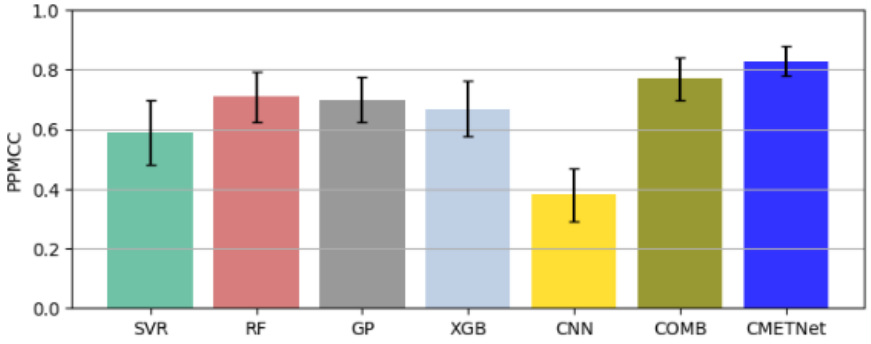
- Plotting the Pretrained Models Results

In [1]:

```
import warnings
warnings.filterwarnings('ignore')

import os
print('Packages imported')
#make sure the scripts are executed in the correct pacakage location.
if os.path.exists('CMETNet_Package'):
    print('Changing working directory to CMETNet_Package..')
    os.chdir('CMETNet_Package')
import sys
sys.path.append('.')

from pretrained_models_results import plot_results
#Plotting the results
plot_results()
```

```
Packages imported
Changing working directory to CMETNet_Package..
```





- CMETNet Training and Testing
    - CNN Model Training and Testing

In [2]:
```
print('Running the CMETNet_CNN_train.py file...')
print('Starting with data preparation, then training the CNN model of CMETNet')

%run CMETNet_CNN_train.py
```

```
Running the CMETNet_CNN_train.py file...
Starting with data preparation, then training the CNN model of CMETNet
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 256, 256, 64)      7808
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      495680
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 64)      256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 128)     991360
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 128)     512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 128)       1982592
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 128)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 256)       3965184
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 64, 64, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 256)       7930112
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 256)       1024
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 262144)            0
_________________________________________________________________
dense (Dense)                (None, 1024)              268436480
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1025
_________________________________________________________________
activation (Activation)      (None, 1)                 0
=================================================================
Total params: 283,813,569
Trainable params: 283,811,905
Non-trainable params: 1,664
_________________________________________________________________
```

```
Epoch 1/10
4/4 [==============================] - 110s 26s/step - loss: 681.5511 - val_loss: 5468.3667
Epoch 2/10
4/4 [==============================] - 99s 24s/step - loss: 548.7322 - val_loss: 1006.2743
Epoch 3/10
4/4 [==============================] - 99s 24s/step - loss: 725.4680 - val_loss: 1457.2574
Epoch 4/10
4/4 [==============================] - 98s 24s/step - loss: 719.3953 - val_loss: 1382.3331
Epoch 5/10
4/4 [==============================] - 96s 23s/step - loss: 726.6939 - val_loss: 606.1398
Epoch 6/10
4/4 [==============================] - 97s 24s/step - loss: 424.7784 - val_loss: 387.8782
Epoch 7/10
4/4 [==============================] - 98s 23s/step - loss: 270.4783 - val_loss: 333.1194
Epoch 8/10
4/4 [==============================] - 96s 23s/step - loss: 151.9844 - val_loss: 223.7884
Epoch 9/10
4/4 [==============================] - 97s 24s/step - loss: 104.8791 - val_loss: 42.8680
Epoch 10/10
4/4 [==============================] - 97s 23s/step - loss: 71.7454 - val_loss: 29.5061
Done ------------------------------------
```

- COMB Model Training and Testing

In [3]:
```
print('Running the CMETNet_COMB_train.py file...')
print('Starting with data preparation, then training the COMB models of CMETNet')

%run CMETNet_COMB_train.py
```

```
Running the CMETNet_COMB_train.py file...
Starting with data preparation, then training the COMB models of CMETNet
Epoch  01 /10
Epoch  02 /10
Epoch  03 /10
Epoch  04 /10
Epoch  05 /10
Epoch  06 /10
Epoch  07 /10
Epoch  08 /10
Epoch  09 /10
Epoch  10 /10
Done ------------------------------------
```

- Testing the Ensemble Learning method
  - Combining CNN and COMB models to show CMETNet prediction results

In [4]:
```
print('Running the results_ensemble.py file...')
print('Final results also will be stored in the results directory')
%run results_ensemble.py
```

```
Running the results_ensemble.py file...
Final results also will be stored in the results directory

       Observed Data                              Prediction
CME Appearance Time    CME Arrival Time             CMETNet Difference in Hours
2014-01-04 21:25:00 2014-01-07 14:25:00 2014-01-07 07:14:12                   7
2014-01-07 18:24:00 2014-01-09 19:24:00 2014-01-09 22:19:48                  -2
2014-01-14 10:24:00 2014-01-18 17:24:00 2014-01-17 04:26:24                  36
2014-01-30 16:24:00 2014-02-02 22:24:00 2014-02-02 03:10:12                  19
2014-02-04 01:25:00 2014-02-07 15:25:00 2014-02-06 16:55:36                  22
2014-02-12 06:12:00 2014-02-15 13:12:00 2014-02-15 07:46:48                   5
2014-02-18 01:25:00 2014-02-20 02:25:00 2014-02-20 14:52:36                 -12
2014-02-19 16:00:00 2014-02-23 06:00:00 2014-02-22 17:20:24                  12
2014-02-25 01:25:00 2014-02-27 20:25:00 2014-02-27 03:20:48                  17
2014-03-23 03:48:05 2014-03-26 04:48:05 2014-03-25 14:19:17                  14
2014-04-01 17:00:00 2014-04-05 09:00:00 2014-04-04 10:39:36                  22
2014-04-02 14:00:00 2014-04-05 22:00:00 2014-04-05 00:38:24                  21
2014-04-08 23:12:00 2014-04-11 08:12:00 2014-04-11 14:50:24                  -6
2014-04-18 13:25:51 2014-04-21 06:25:51 2014-04-20 20:59:27                   9
2014-06-25 12:54:00 2014-06-29 17:54:00 2014-06-28 07:07:12                  34
2014-08-15 18:12:00 2014-08-19 06:12:00 2014-08-18 12:00:00                  18
2014-09-09 00:16:00 2014-09-12 09:46:00 2014-09-11 15:29:48                  18
2014-09-10 18:24:00 2014-09-12 15:24:00 2014-09-13 01:37:48                 -10
2014-11-01 05:12:00 2014-11-03 22:12:00 2014-11-03 06:07:48                  16
2014-11-07 18:08:00 2014-11-10 01:08:00 2014-11-10 08:20:00                  -7
2014-12-17 05:00:00 2014-12-21 11:30:00 2014-12-20 04:14:24                  31

CMENet PPMCC:    0.62
CMENet MAE:      16.51
Done ------------------------------------
```

References:

- https://wiki.hpc.arcs.njit.edu/index.php/MinicondaUserMaintainedEnvs
- https://realpython.com/python-virtual-environments-a-primer/
- https://stackoverflow.com/