Team 16
Senior Design Project

Group Report

Purpose of Software

The purpose of our software is to solve Lockheed Martin's issue of unoptimized server uptime. They claim that servers sometimes go unnoticed and do not perform optimally since they rely on real people to turn them on and off. This is costing the company potentially high costs for their cloud compute infrastructure. What they wanted was for us to train models for the purpose of forecasting future server loads, that way they could automatically turn the servers on or off in order to bear the forecasted load, instead of having humans do it manually when the need arises. We also included GUI, additional metrics with anomaly detection and clustering, and multiple models for the same purpose of forecasting.

Design of Software

Consisting of a collaborative effort between all team members, the Flask-based GUI website combines algorithm functionality and web-design features to create an easy-to-use and interactive interface for testing our algorithms and gaining a greater understanding of what each algorithm does, how it works, and how it ties into solving the engineering problem we were provided. At its core, our webpage utilizes a web.py file that orchestrates the integration of our algorithms and the back-end rendering of dynamic plots, scheduling, and facilitating the startup of the local host webpage.

List of Functionalities

1. Algorithm Analysis
    a. LSTM Model
    b. Prophet Model
    c. Seasonal Decomposition Model
2. Clustering Algorithm
3. Anomaly Detection

Use Case Scenarios

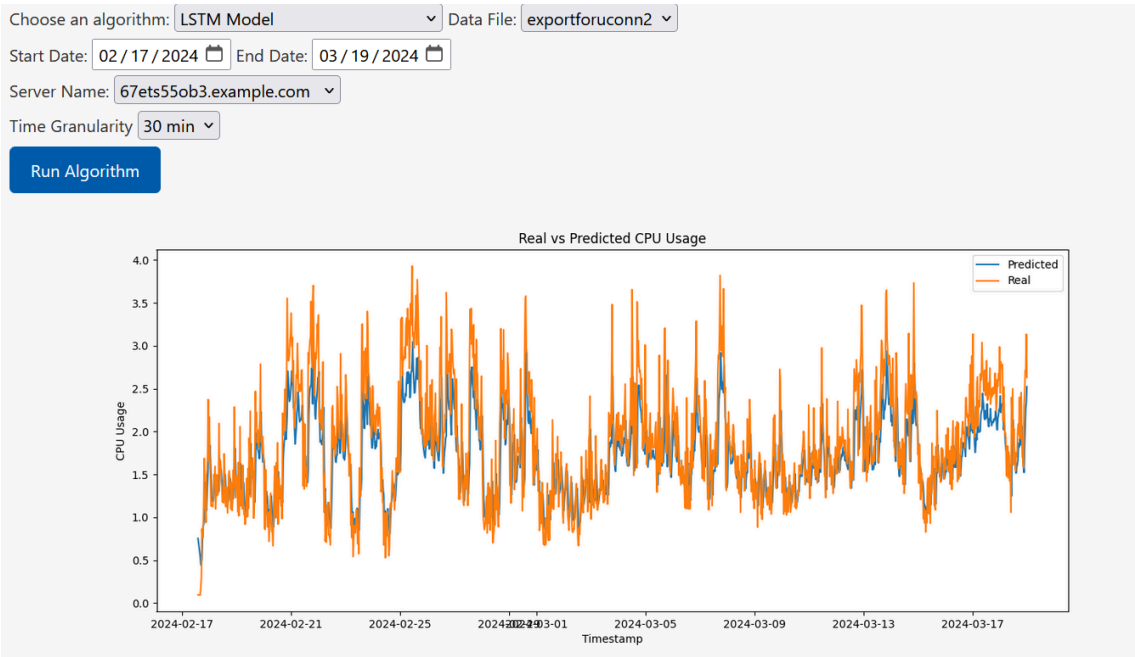
front end and back end

1) LSTM Model

The LSTM model takes as input a csv file of splunk data. It then transforms this data into a format usable by the model, in the process aggregating the servers by timestamp and aggregating the timestamps by a chosen period (half-hour by default). Then the dataset is prepared further, and then the model is trained on the data. By default, we train on 75% of the data and use the remaining 25% as testing data. After evaluating the model on the testing data, a graph showing both the real values of the testing data and the predicted values for those timestamps is output. Additionally, if one wishes to view a schedule for an individual server, the name of that server can be input before the first stage of editing and only information on that single server will be used. Then, after training the model, a schedule can be displayed showing whether it should be turned on or off at a given time.

Front End:

Using and viewing the model in the front end is simple. Simply select the model from the "Choose an algorithm" drop down menu, and select the data file you wish to test on (the model will always be trained on the first file). Next, select a start and end date for the section of the data you wish to view. Select a "Time Granularity" to choose by what period the data should be aggregated by. If you wish to see the model evaluated on all the data then make sure "Server Name" is set to None. Then click the "Run Algorithm" button and the model will train. Note that the model does take ~2.5 minutes to finish training, unlike the Prophet and Seasonal Decomposition models which train relatively quickly. This is because every time the model is run, the csv must be parsed and aggregated (~50 seconds) and then the model must be re-run on this chosen dataset (~90 seconds). Then below, if you wish to see a schedule, you can select the specific threshold you want to use and see when servers should be turned on or off. If you instead wish to view the schedule for a single server, then select an available server from the "Server Name" dropdown. This will also train the model on that one single server and display the schedule for what times that server should be turned on or off, though the model displayed will still be the one that was trained on all data.



Back End:

The backend begins with the data cleaning functions. editor1() takes in the original raw csv and turns it into a pandas dataframe, extracting the relevant features and aggregating all servers by timestamp. If a server name is input, then it will disregard all information not related to that server. Then we aggregate again in the half_hour_agg() function, which aggregates the file output by editor1() by a 30 minute period. There is another function, aggregate_data(), which is functionally the same except it takes as input a period (in the format of a pandas datetime period alias) to aggregate by. The python files new_editor1, half_hour_agg, and input_time_agg all do the same thing. We then define a scalar so that the data can be scaled from 0-1. Then we prepare the data to be used by the model, most notably we transform the data from a list of CPU values to a list of subsequences (lists) of CPU values. For example, data_X[i] is a list of CPU values from timestamps[i-12:i], and data_y[i] is the CPU value associated with timestamps[i]. The LSTM model uses the 12 values preceding CPU[i] to predict CPU[i], this is (part of) the method by which LSTM learns patterns in the data. We then train the model (using MSE as the loss function and Adam as the optimizer). After training, we then test the model on the remaining test data, plotting both the real CPU values and the predicted CPU values. The RMSE of the model is approximately .432, which is ~9.68% of the range of the data, indicating a fit of the data that is accurate but not overfit. In order to view a schedule for the data, we run the model again but specify a single server which we want to view a schedule for. We then find the mean and standard deviation of the CPU values for that server and create upper and lower bounds for the server. Then we predict the values of a server for a given day and set the server to turn off once its predicted CPU value drops below the lower bound (mean-std) and turn on once it exceeds the upper bound (mean+std), with the initial value being set to off unless the CPU value is greater than the upper bound. It is worth noting that it is very simple to evaluate the model on separate

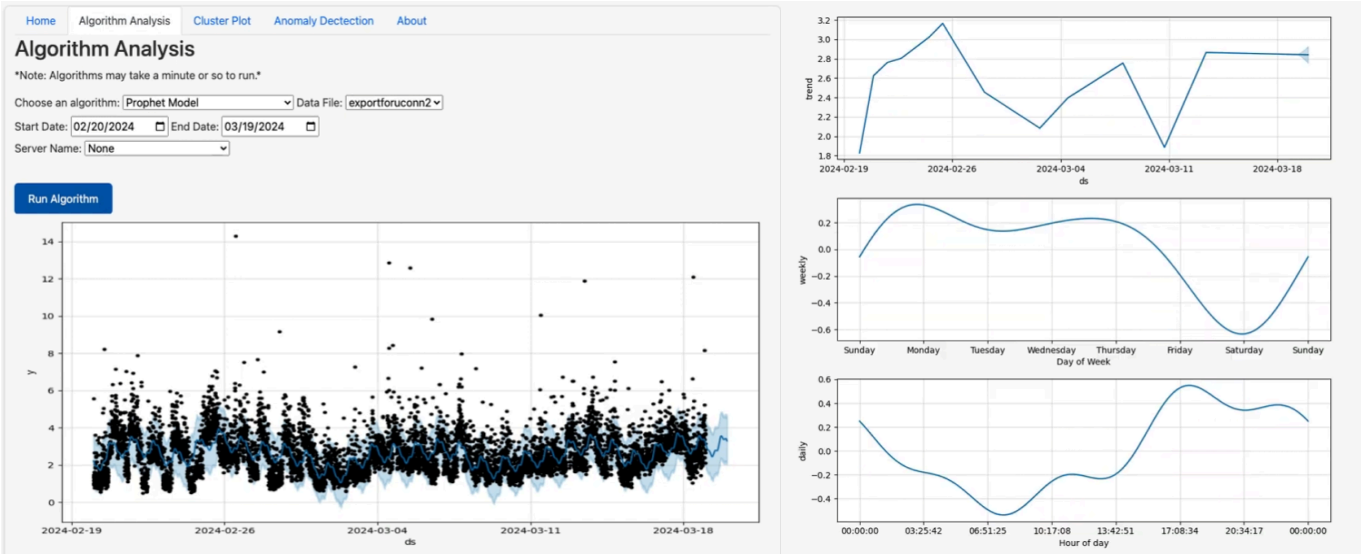
datasets, simply run the data cleaning functions but be sure to specify in the train_test_split in prepare_data() to 0, and then run the Testing function with the model and the test loader.

2) Prophet Model

Prophet's prediction process begins by taking in historical performance metrics (Splunk data in this instance) for a collection of servers. Historical data is then manipulated according to user specification, and the Prophet model then fits each of its components (Trend, Seasonality, and Holidays) to the manipulated data. After fitting, the model is given a length of time in which to forecast, and makes predictions on 95th percentile CPU usage at 5 minute intervals over the specified time frame. After concluding running, the model will display the overall trend, along with each available component of the fit (components are made available via the length of time offered in the original data – i.e. a dataset that offers years of historical data will have a daily, weekly, monthly, and yearly component, whereas a dataset with only a few months of data will not have a yearly component).

Front End:

After selecting the Prophet model and preferred dataset from the dropdown menus, along with a start and end date for visualization, click the "Run Algorithm" button to begin the prediction process. The model will train on an aggregation of the data from the servers in the chosen data file, and then provide a visualization of the overall trend, followed by each available



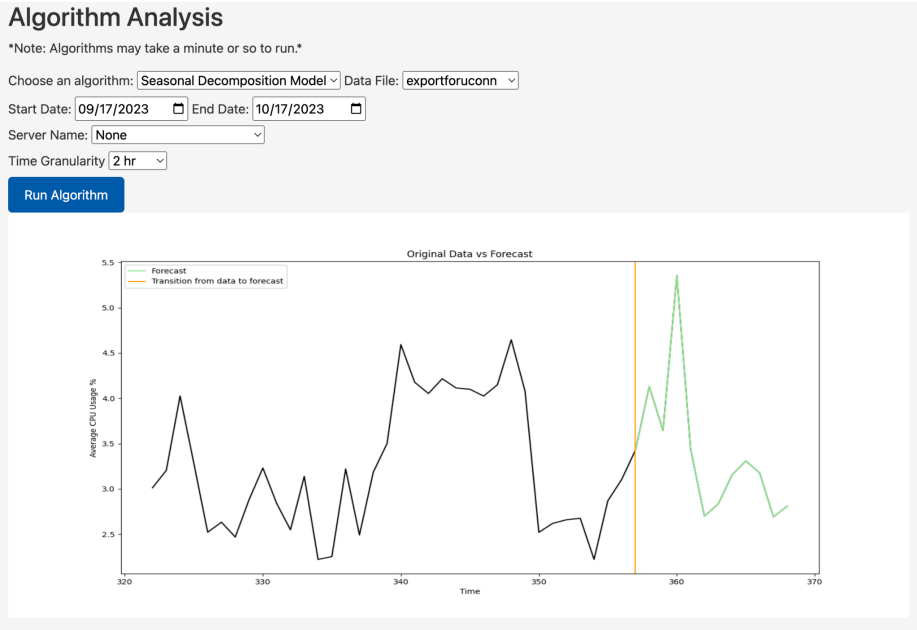
component. This information is incredibly useful for determining the repetitive nature of the data over different scales of time. For businesses like Lockheed Martin which operate around a 4 day

workweek, there is a noticeable drop in weekly trend data starting on thursday and picking up again on monday. Following these graphs, is the scheduling section. Here, a threshold is automatically populated based upon the median of the future predictions made by the model. A schedule is then output which determines scaling up and scaling down periods for the overall group of servers, based on their relation to the threshold. This threshold can be modified by the user to change the scheduling behavior as they see fit. The one function that hasn't been mentioned yet is the "Server Name" dropdown. If a server name is selected, the data preparation step isolates the metrics for the specific server selected, and provides this to the model for training instead of an aggregation. Each graph will then be updated according to only the individual server selected, and allows for the scheduling section to provide a more individualized On/Off schedule for the given server.

Back End:

For the back end, we can begin by looking at the data preparation step. It begins by isolating servers which are determined to be inactive based on their 0 average CPU metric. These servers are then excluded from the Pandas DataFrame containing the input data. In the instance where a "host_name" is specified, the DataFrame is populated with the 95th percentile CPU data for the server corresponding to the given host. Otherwise, the dataset is populated with the average CPU data over all the servers, aggregated into 5 minute intervals. The df is prepared for the model as soon as it is in the form of 2 columns (date stamp 'ds' and the data point being considered 'y'). This dataframe is then fed into the model to be fit, and another dataframe is created to specify the forecast horizon, or the time frame over which to make predictions. This dataframe is set up in 288 five minute intervals (24 hours in 5 minute intervals). The prediction is given in a dataframe that contains the date stamps, y_hat predictions, and upper/lower bounds for those predictions. From here, the future predictions are isolated and then used to obtain a median for scheduling purposes.

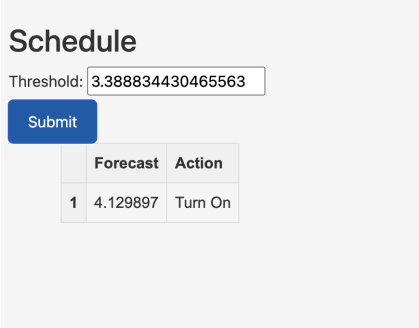
3) Seasonal Decomposition Model



Front End:

The use of the model is similar to that of the other models. In the image above the dashboard for the seasonal decomposition model is presented. We walk through how to use it and make use of its customizations. To select this model, first click the dropdown option of "Seasonal Decomposition Model" under "Choose an algorithm:". Next, we have the "Data File" field, here you insert the server log file, the expected file is a CSV with at least fields corresponding to a host, timestamp and 95th Percentile CPU usage, these are the minimum fields necessary for usage. Next, we have the "Start Date" and "End Date" fields, these correspond to the time window which you want to capture from the data. Next, we have the "Server Name" field, here, you have the option to select an individual server name from the data file, but if you want to look at the data for all servers, selecting the "None" option includes all server data. The Last field we must consider is the "Time Granularity" this option enables the user to choose the granularity within which they view the data, this is to say, we can consider the data in its native form, the 5 minute interval, or we can scale up to see the 1 hour, 2 hr,24 hour intervals and more. Finally, the user can click "Run Algorithm" and a plot is generated. This plot corresponds to the original data, and the models forecast. The distinction is made by the vertical yellow bar, where the plot transitions from real data, into the green, which is the model forecast. If we look again at the top image, we see that this was the result of running our model on the exportforuconn data set, on the entirety of its time range, with all servers selected, and a time granularity of 24 hours.

Below the forecasting model, we have the scheduling component.



Here, the results entered from the top of the pages data fields are saved, and a schedule is output representing the recommended Times that server(s) should be scaled up or down depending on the forecasted usage.

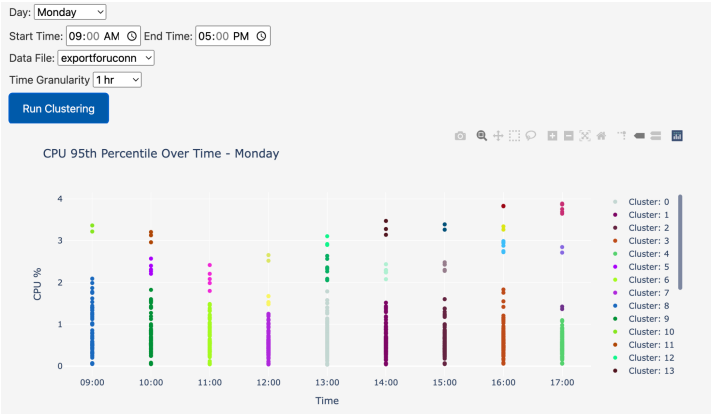
Back End:

On the back end, we start off with some scripts that are useful for preparing data for the model as well as adjusting the data by the specifications provided by the user, we have datacleaning.py which takes care of creating smaller CSVs with only the most relevant data specified by the user, we track "inactive" (defined as having 0 usage over time) servers, and only include helpful server data in the outputted csv, in addition, we average our data out for use in tracking data across all the servers if requested. Next, we have time_splitter.py, which does the job of breaking our data into the correct time granularity as specified by the user, this outputs the usable csv by our model. After loading in the cleaned CSV's, we delete any created files to keep the directory clean, and for easy reusability. Next, our script feeds this dataframe into our model object and the model gets trained and forecasts are generated. The number of forecasts is dynamic with the time granularity, but we aim to be able to forecast at minimum 24 hours for low timeframes, and about 14 forecasts for high time frames such as the 1 day granularity. Next, we generate the plots and the schedules, for the schedules, we take our mean forecasted value and create an upper bound by adding a standard deviation of our forecasts, next we create a lower bound in a similar fashion, but we decrease our mean by a standard deviation, We create an action when our forecasted usage is above our upper bound or below our lower bound.

4) Clustering Algorithm

The clustering algorithm takes in a dataset (and other additional features) and outputs a plot which groups together servers based on how closely related their CPU usage values are. It then uses this plot to create a schedule of servers which is used to tell a user when certain servers/groups of servers should be on or off at a given time. This can be used in scenarios where a company has a lot of servers and are not allocating its resources appropriately. Using this software it will come up with a precise schedule telling this company exactly which servers

should stay on or off at a given time. For example, as pictured below, I used the algorithm on LockHeed Martin's dataset and received the below schedule of servers that should be on at a certain time. For instance, we see that group 10's servers should be on at 9:00 am.



Server	group 0	group 1	group 10	group 11	group 12	group 13	group 14
Time							
09:00	Never Used	Never Used	On	Never Used	Never Used	Never Used	Never Used
10:00	Never Used	Never Used	Never Used	On	Never Used	Never Used	Never Used
11:00	Never Used	Never Used	Never Used	Never Used	Never Used	Never Used	Never Used
12:00	Never Used	Never Used	Never Used	Never Used	Never Used	Never Used	Never Used
13:00	Off	Never Used	Never Used	Never Used	On	Never Used	Never Used
14:00	Never Used	Off	Never Used	Never Used	Never Used	On	Never Used

Front end:

In order to use the front end part of the clustering algorithm a user would start by picking from the dropdown menu of days, Mon-Sun, which will show results for that chosen day. The user will then pick a start/end time from the time menu which will show results from that starting time to that ending time. The user will then be asked to choose from a list of files, each containing different data. Lastly, the user will be asked to pick from a list of time granularities, which will group the results in time intervals specified by the user. After filling out this information the user will press the "Run Clustering" button, and the plot produced by the clustering algorithm will appear. Next, the user will type in a threshold value that will be used to create the schedule for each group of servers. At first the input value will automatically be filled with the mean CPU usage value across all clusters. The user will then either continue with the given value or choose a new one and press the submit button and a schedule for each group of servers will appear. Lastly, the user can look at which group (column) and time (row) the schedule states that servers should be on or off. The user will be prompted to submit a group number and after pressing the submit button will see the list of server names in that group displayed in a table.

The only thing you should need to change in the front end is given to you as input values, however if you wanted to add another option to one of the dropdowns, for example, the datafile you would need to add it to your current directory, go to the web.html code and add the filename under the available options using the line <option value= "filename">filename</option>. For other options such as time_granularity you would add the new time under the other available options using the same line. Other than this there should be nothing else that you need to change/add/remove in the front end.

Back end:

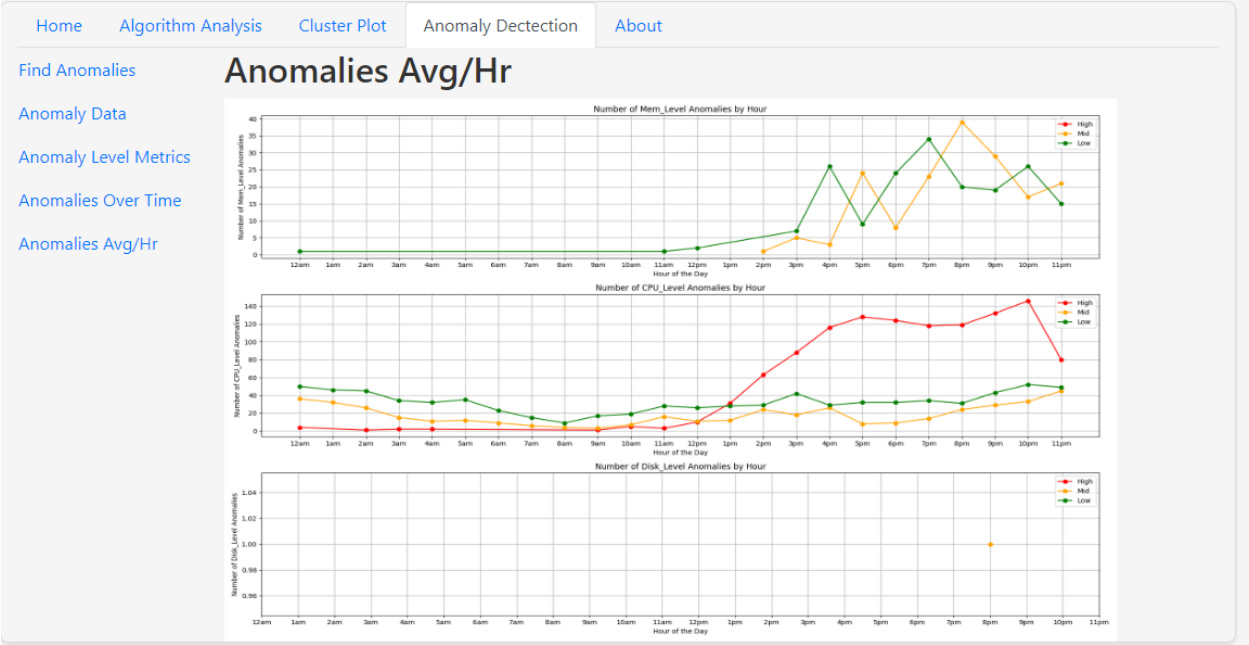
In order to use the back end part of the clustering algorithm you would first input the name of the file that you would like to run the clustering algorithm on (make sure that the file you choose is in the directory that you are running the file in). It will then ask you to input the time granularity you would like to see and it will create some files after doing the appropriate data cleaning using the inputs that you chose. Then type in a day of the week, Mon-Sun, time range in the format (00:00), and then a plot will appear in your browser. After viewing the cluster plot if you would like to see the schedule, you will go back to your terminal and submit a threshold, to which it will show you the recommended value (the median CPU value across all clusters), and will then output a schedule based on if a certain cluster is below or above the threshold you choose (if above the server is stated as on, if below the server is stated as off). Finally, it will ask you if you want to view a certain group number and you will simply type in that number to view the individual servers inside each cluster/group.

The only things you should need/want to change within this code are offered to you as inputs. If you would like to add something like another csv file and get different results from different data, all you would need to do is add it to your directory and use it as one of your input arguments.

5) Anomaly Detection

Front end: My software simply detects erroneous data based on the data file that is selected on the website. You can also select the server name that you want to analyze. Since a lot of the servers never actually turned on or had too many missing values, there are not many servers to choose from. Though you will see that by running my algorithm on the servers, multiple charts are generated through the side categories. If you run my application with server name set to none, it will process all of the servers and instead provide a compilation of all of the metrics, and it will also provide which server had the most issues. If graphs appear as empty, or bars are missing, that means that there were no anomalies tracked. The three metrics I tracked were memory, disk usage, and cpu usage. All three of these metrics individually get tracked for anomalies. Anomalies are determined in my software by thresholds, there are three of them. Thresholds include low, medium, and high. Within the code these thresholds can be changed, right now they are set to something like 70% usage for low, 80% usage for medium, and 99% usage for high. If the metrics ever fall within the thresholds they are tracked by my software. As for a use case, my software can find servers which are obviously not performing optimally, since a server with over 1000 logged instances of being above 99% usage likely should raise some concerns. It is considered anomalous since Lockheed was very clear about wanting servers to have balanced loads not exceeding certain thresholds. I explain the reasoning behind my threshold approach in my limitations section, considering there are many approaches to anomaly detection. Finally, the graphs I provide are described by the subcategories on the website, and on

the Google Colab, but it includes time series information, highest anomaly information, number of anomalies and types of anomalies, etc..



Back end: The individual files of mine can be simply ran through Google Colab, all they require is for the server data to be within the directory files and for the correct filename to be set at the top of the code block. The functionality is basically exactly the same, except now the software will ask for you to input host names manually if you would like to look at an individual server.

Limitations

1) Website

The website is limited by the fact that the current framework can only handle a certain level of computational complexity and functionality before it would begin to slow and lose efficiency. Due to the extensive nature of machine learning computations and large data sets, it would eventually get outclassed and not be able to handle newly integrated functionalities as seamlessly, affecting the usability and user-experience of the webpage. But, there is an array of technologies and methods that can be utilized to enable the backend to be able to handle heavier computational loads.

2) LSTM Model

The biggest limitation of the LSTM model is the fact that the only features from the dataset that are used in the model are timestamps and the 95th percentile CPU. Our concern was determining the load on individual servers, and believed that this one feature was the most

relevant one in terms of determining load on the server. However, we may have been naive in ignoring other features such as CPU max, disk average, and memory, as those features could have also had an impact on server load as well.

3) Prophet Model

One of the main limitations of the current implementation is the lack of consideration of other variables, and the nature of the scheduling. Since the scheduling is currently based on thresholding, it can easily be swayed by outliers (although we attempt to mitigate this impact by utilizing the median as a basis). If the schedule were dynamically computed utilizing the components of the model as a guide, the schedule could more intelligently adapt to unstable trends. The model can also suffer in instances where there isn't enough training data.

4) Seasonal Decomposition Model

When considering this model, the greatest limitation is that the time series model is univariate, meaning that the model can only use and predict a single feature from a dataset. In context, this means that our model can only consider one of the variables provided in the dataset, we chose 95th percentile CPU, but others such as disk avg or CPU max could have been used for example. This is a limitation, because there are other temporal relationships that could exist in the data between different features that could create a more robust model, but with this framework those are unable to be acted upon.

5) Clustering Algorithm

The limitations of this clustering plot is that I was unable to choose the best epsilon value and minimum group value for each plot produced by the clustering algorithm. Meaning that no matter what inputs the user gives, these same values will be responsible for determining how all servers are grouped.

Thus if I had more time I would like to be able to go through all options that a user can choose and set appropriate values that give me better clustering results for each variation of inputs that a user can give to the clustering algorithm.

6) Anomaly Detection

Limitations of anomaly detection had to do with the data we were given. Since most servers were missing a lot of logged information as NaN data, and because we were not given a very long period of time of server usage, it was quite difficult to use more complicated statistical methods like z-score or machine learning since that relies on anomalies determined by average

performance. Since there was not much data to find stable averages from, the results from using these more advanced algorithms were very unpredictable and often incorrect. The fact that computer metrics are very volatile also was not helping. So instead, I settled with the more simple threshold solution.

Future Improvements

1) Website

For future improvement of the website framework, I could implement an updated design that utilizes distributed computing frameworks and/or cloud-based infrastructure. Additionally, working with team members to refactor code and simplify functionality would improve computational efficiency.

2) LSTM Model

A potential future improvement for my model would be changing the structure of the LSTM model itself. I decided to use a simple LSTM model, but other more advanced variants do exist, namely the Stacked LSTM and Bidirectional LSTM. I decided not to use them because the amount of data I received to train on was limited, and I thought these more advanced models would overfit the data. However, if Lockheed Martin wishes to use my model and instead train it on a larger dataset that they procure, it may be worth their time to switch the model to a Stacked or Bidirectional LSTM model instead. Additionally, the way my model is built makes it such that it does not predict past the last timestamp in the test dataset. This is because I wanted to prioritize showing how the model predicts cpu usage compared to the real cpu usage in order to show how accurate the model is. Forecasting past the end of existing test data does not help to show how accurate the model is since there is no ground truth to compare to. However, this would be a useful feature for the sponsors. Thankfully, it would be easy for the sponsors to implement, all that would need to be done is modify the Testing function to iterate beyond its current limit, which is the end of the given testing data.

3) Prophet Model

As mentioned in the previous section, additional regressors could be utilized to provide the model with more context of the data. Also, including more precise tuning of the hyperparameters into the final pipeline could increase training time, but potentially provide increased accuracy. And again, the implementation of a more dynamic scheduling system would be beneficial to handle scheduling of individual servers. To stretch these ideas further, the model could be trained on each server individually and then output a schedule composed of each individual server's predictions.

4) Seasonal Decomposition Model

There are a couple future considerations in mind that could make this a more robust modeling framework. As we hinted at in the limitations, looking at multivariate models could prove helpful as being able to use multiple variables for a forecasting model could prove to be more useful as we can consider more of our data. In addition, for our scheduling component, we could look at better fine tuning the decision rule with testing data to find what type of decision rule proves to lead to most efficient server usage.

5) Clustering Algorithm

As mentioned above I would like to be able to change certain variables specified within the code that are responsible for producing different outputs based on user input such that no matter what input the user gives, the best clusters for that variation will be formed.

6) Anomaly Detection

Future improvements would include using more advanced statistical methods to find anomalies, this could be done easily using more fitting data. All of the metrics and graphs I have should still function if algorithms are replaced correctly. This may provide more reliable results for whatever Lockheed is looking for.