Abstract

Cybersecurity has become an essential aspect of every modern business. Targeted cyberattacks can have catastrophic consequences, not only for businesses but also for governments and daily life. For instance, let's consider the electric power grid. If the Western Interconnection power grid were to be attacked, it could result in a power outage affecting British Columbia, Alberta, and the western United States. To minimize the risk of such disasters, protecting critical infrastructure is crucial. In our project, we aim to practice Critical Infrastructure Protection by utilizing unsupervised intrusion detection through time series analysis and forecasting, applied to streaming data from a supervisory control system.

Table of Contents

Technical Background

In August 2003, the Eastern Interconnection electric power grid suffered a cascading failure due to an attack, affecting all of southeastern Canada and eight northeastern U.S. states, resulting in a power outage for 50 million people for two days. This event inspired our project, which focuses on anomaly detection in electric power grids. The three main steps we will follow are feature engineering using Principal Component Analysis (PCA), training and testing a Multivariate Hidden Markov Model (HMM), and finally, anomaly detection. Our data consists of one main dataset and three anomaly detection datasets, with parameters including Date, Time, Global_active_power, Global_reactive_power, Voltage, Global_intensity, Sub_metering_1, Sub_metering_2, and Sub_metering_3.

Figure 1 below shows an example of our data:

```
"Date","Time","Global_active_power","Global_reactive_power","Voltage","Global_intensity","Sub_metering_1","Sub_metering_2","Sub_metering_3"
"16/12/2006","17:24:00",NA,0.418,234.84,18.4,0,1,17
"16/12/2006","17:25:00",5.36,0.436,233.63,23,0,1,16
"16/12/2006","17:26:00",NA,0.498,233.29,23,0,2,17
"16/12/2006","17:27:00",5.388,0.502,233.74,23,0,1,17
"16/12/2006","17:28:00",NA,0.528,235.68,15.8,0,1,17
"16/12/2006","17:29:00",3.52,0.522,235.02,15,0,2,17
"16/12/2006","17:30:00",3.702,0.52,235.09,15.8,0,1,17
"16/12/2006","17:31:00",3.7,0.52,235.22,15.8,0,1,17
"16/12/2006","17:32:00",3.668,0.51,233.99,15.8,0,1,17
"16/12/2006","17:33:00",NA,0.51,233.86,15.8,0,2,16
"16/12/2006","17:34:00",4.448,0.498,232.86,19.6,0,1,17
"16/12/2006","17:35:00",NA,0.47,232.78,23.2,0,1,17
"16/12/2006","17:36:00",NA,0.478,232.99,22.4,0,1,16
"16/12/2006","17:37:00",NA,0.398,232.91,22.6,0,2,17
"16/12/2006","17:38:00",NA,0.422,235.24,17.6,0,1,17
"16/12/2006","17:39:00",3.384,0.282,237.14,14.2,0,0,17
"16/12/2006","17:40:00",NA,0.152,236.73,13.8,0,0,17
"16/12/2006","17:41:00",NA,0.156,237.06,14.4,0,0,17
"16/12/2006","17:42:00",3.266,0,237.13,13.8,0,0,18
"16/12/2006","17:43:00",NA,0,235.84,16.4,0,0,17
"16/12/2006","17:44:00",5.894,0,232.69,25.4,0,0,16
"16/12/2006","17:45:00",NA,0,230.98,33.2,0,0,17
"16/12/2006","17:46:00",NA,0,232.21,30.6,0,0,16
"16/12/2006","17:47:00",5.174,0,234.19,22,0,0,17
```

Figure 1

Feature Engineering

After importing the dataset using read_csv(), we prepared it for PCA by removing any NA values and non-numerical values using na.omit(). We also eliminated the non-numerical columns (date and time) by removing the first 2 columns of the dataset.

PCA is a technique that reduces dataset dimensions by identifying the most important features that contribute to the most variance. By focusing on response variables with the highest impact on dataset variation, PCA transforms a large set of variables into a smaller set of principal components while maintaining most of the original information. It also enables us to visualize data spread and determine the percentage of variance each component contributes.

To perform PCA, data normalization is necessary as PCA identifies variables with the largest variances. Normalization ensures all variables have the same relative influence by scaling them to a mean of zero and a standard deviation of 1. We used the scale() function to normalize the data by applying it to all numeric data with the following code:

num_data <- num_data %>% mutate_all(~(scale(.) %>% as.vector))

We computed the correlation matrix with the cor() function and plotted it using ggcorrplot() and that is seen in figure 2 below:

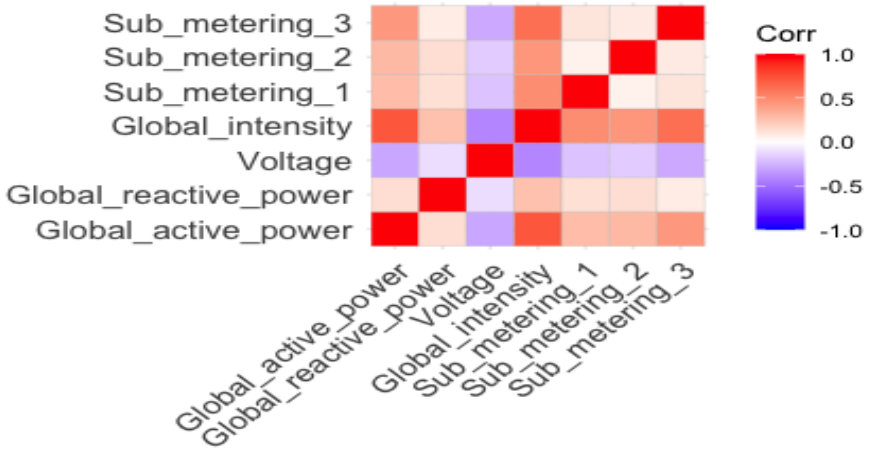


Figure 2

The results of the correlation matrix in figure 2 help us visualize the following:

- The closer the values to 1, the more positively correlated the 2 variables are.
- The closer the values to -1, the more negatively correlated the 2 variables are.

We performed PCA analysis by calling princomp() on the correlation matrix generated from the previous step. The summary() function was then called to obtain the results, as seen in figure 3 below:

```
Importance of components:
                          Comp.1    Comp.2    Comp.3     Comp.4     Comp.5      Comp.6
Standard deviation     0.7966595 0.3621709 0.3554087 0.31512249 0.18535387 0.071042821
Proportion of Variance 0.6156700 0.1272418 0.1225346 0.09632994 0.03332774 0.004896017
Cumulative Proportion  0.6156700 0.7429118 0.8654463 0.96177625 0.99510398 1.000000000
                             Comp.7
Standard deviation     4.533478e-09
Proportion of Variance 1.993726e-17
Cumulative Proportion  1.000000e+00
```

Figure 3

The PCA results show 7 principle components generated, which matches the number of variables in our dataset after removing the non-numerical ones. The "proportion of variance" row indicates the percentage of variance explained by each component. Component 1 accounts for about 61.6% of the variance, and component 2 accounts for about 12.7%. Together, the first 2 components explain around 74.3% of the variance, suggesting that they can represent the data reasonably well.

Finally, the contribution of each variable to the two principal components can be determined by computing the square cosine value, or Cos2. The resulting graph below shows that Voltage and Global_intensity have the highest Cos2 values, indicating that they are good

representations of the variables on those components and contribute the most to the two principal components.

To generate the graph seen in figure 4 below, we used the following:

fviz_cos2(data.pca, choice = "var", axes = 1:2)

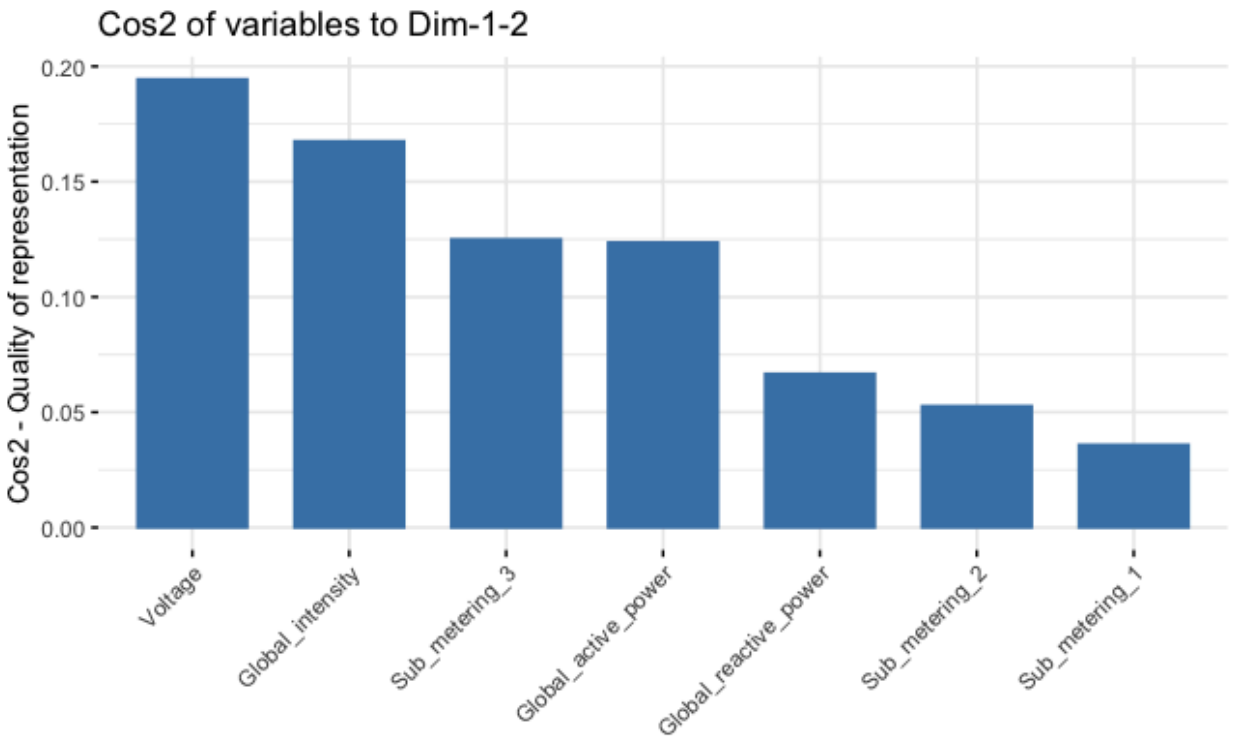


Figure 4

The biplot in figure 5 below shows the representation of variables based on their Cos2 values:

- High Cos2 variables (Global_intensity and Voltage) are in green.
- Medium Cos2 variables (Global_active_power and Sub_metering_3) are in orange.
- Low Cos2 variables (Sub_metering_1, Sub_metering_2, and Global_reactive_power) are in black.

To generate the biplot, we used the following:

fviz_pca_var(data.pca, col.var = "cos2", gradient.cols = c("black", "orange", "green"), repel = TRUE)

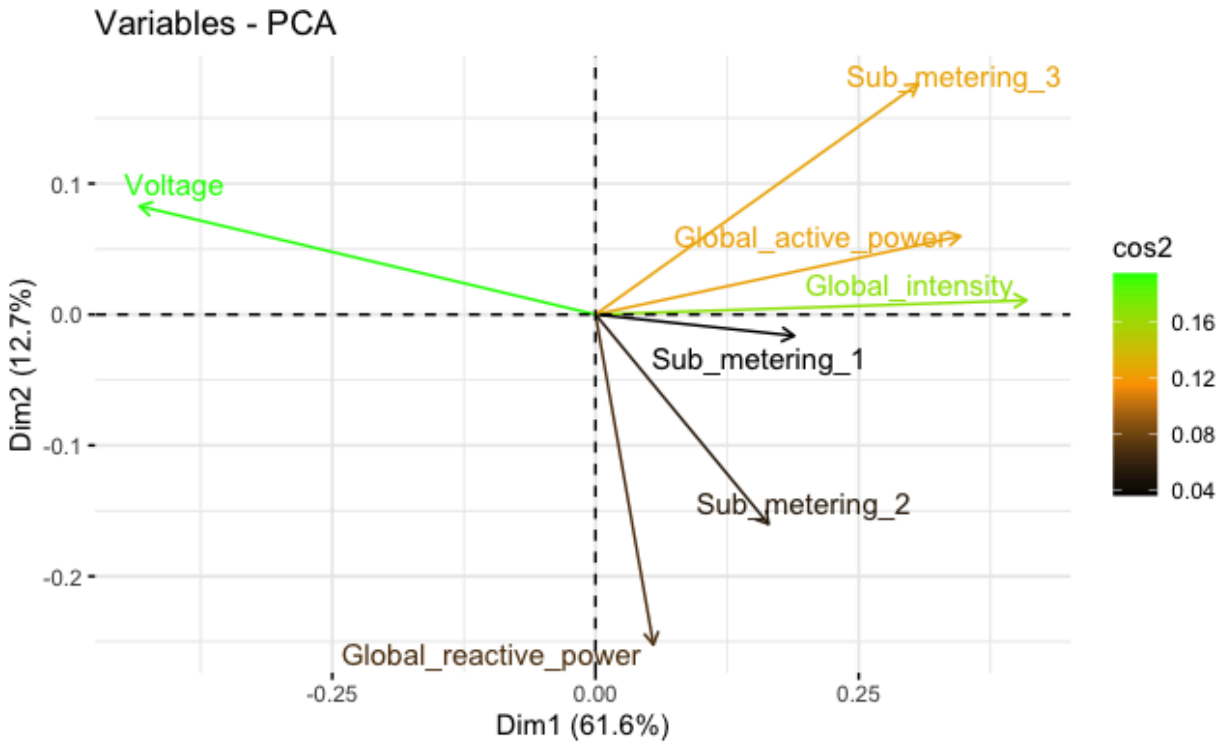


Figure 5

Based on the representation of variables in the biplots, along with their correlation with each other and the dominance of principal components 1 and 2 over others, we have chosen the subset of response variables for our Hidden Markov analysis as **Voltage and Global_intensity**.

HMM Training and Testing

Choose a weekday or a weekend day and a time window between 2 to 6 hours on that day.

We utilized ggplot to create a preliminary visualization of a random sample of Monday's voltage consumption data. The selection of voltage is based on its significance as the most important parameter identified through PCA.

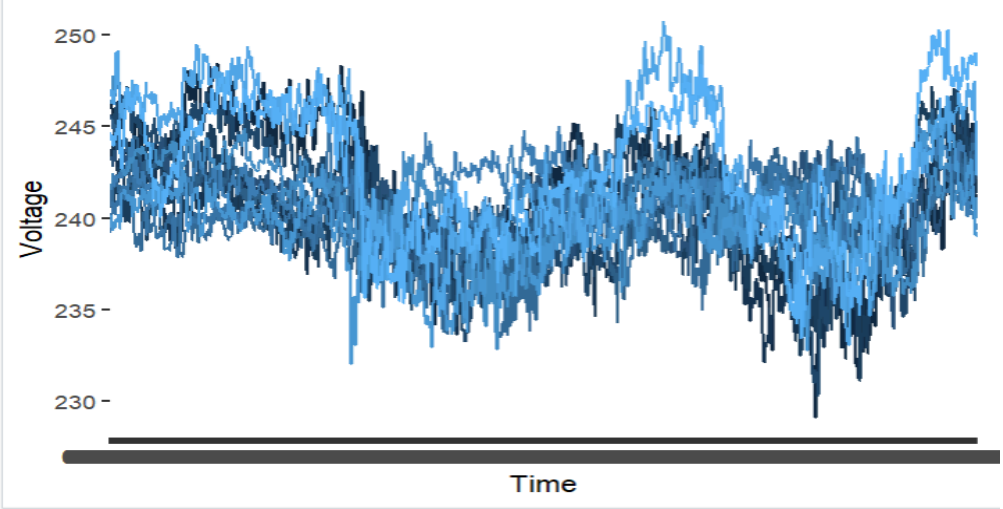


Figure 6

We aim to select a time interval that exhibits a discernible pattern. Upon examining the graph, we observe that towards the end of Monday, the voltage consumption remains steady at around 240V before suddenly spiking to 245V. Thus, we would like to investigate this specific time period further.

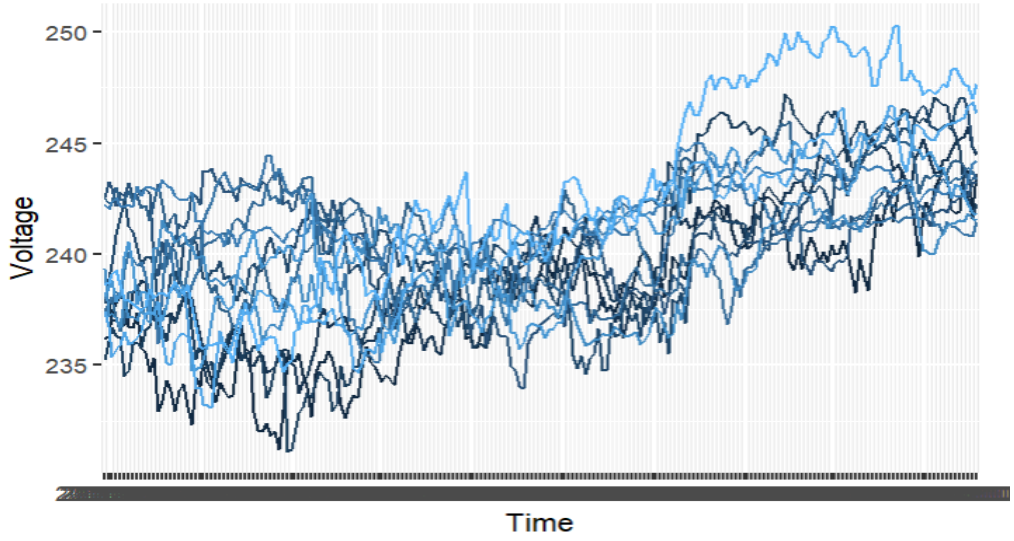


Figure 7

Figure 7 above is a closer look at the timeframe we chose. **Monday 20:00:00 to 23:30:00**.

Selecting the Training and Test Data Sets

After applying the Monday timeframe filter and removing any missing values, we found that our dataset contains 32,705 observations. To train our model, we randomly split the data into an 80% training set and a 20% test set. We chose to allocate more data for training to ensure our model has sufficient data to learn from. The test set, which contains the remaining 20% of the data, is used to evaluate the performance of our model.

train_set <- row[(1:26164),]

test_set <- row[(26165:32705),]

The train_set simply takes the first 80%, and test_set the last 20%.

Multivariate HMM Training

We need to choose a state between 4 to 24 to train our Hidden Markov Model. We started by trying out a low number of states, but we found that the negative log-likelihood value was too large as seen below in figure 8:

```
converged at iteration 161 with logLik: -26952.38
[1] 6
Convergence info: Log likelihood converged to within tol. (relative change)
'log Lik.' -26952.38 (df=59)
AIC:  54022.76
BIC:  54504.91
```

Figure 8

States up to 14 are still large as seen below in figure 9:

```
converged at iteration 161 with logLik: -12413.29
[1] 14
Convergence info: Log likelihood converged to within tol. (relative change)
'log Lik.' -12413.29 (df=251)
AIC:  25328.57
BIC:  27379.78
```

Figure 9

We observed a decrease in log-likelihood during EM iteration for state numbers 19 to 20, indicating potential overfitting. Therefore, state number 19 appears to be a suitable candidate as illustrated in Figure 10 below.

```
[1] 19
Convergence info: Log likelihood converged to within tol. (relative change)
'log Lik.' -6664.552 (df=436)
AIC:  14201.1
BIC:  17764.16
[1] 20
Convergence info: likelihood decreased in EM iteration; stopped without convergen
ce.
'log Lik.' 16235.48 (df=479)
AIC:  -31512.97
BIC:  -27598.51
```

Figure 10

BIC and Log likelihood of the states can be seen in figure 11 below.

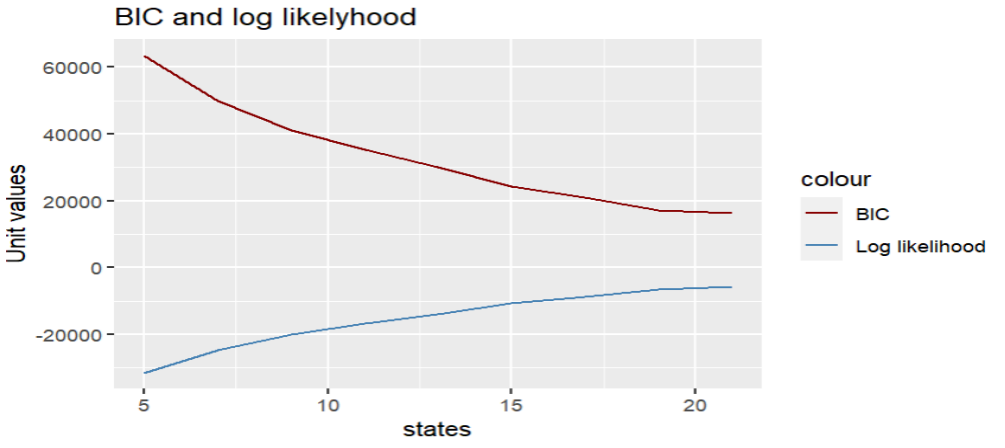


Figure 11

Through some remodelling, we finalize: the **log like of state 19 = -5914, BIC = 16263**

Applying to the Test Set

We use getpars setpars, then run forwardbackward to calculate log-likelihood on the test set.

To get a model for the test set:

mod2 <- *depmix*(#testset, state=19)

To get pars from the fitted trained model, then set pars to the new test model:

mod2 <- setpars(mod2, getpars(fm1))

fb<-forwardbackward(mod2)

fb$logLike

```
> fb$logLike
[1] -4487.744
```

Figure 12

As shown in Figure 12 above, the output log-likelihood of the test model is better than that of the trained model, with a value of -5914. Therefore, we can conclude that our model is suitable for anomaly detection.

Anomaly Detection

For anomaly detection, we calculate the log-likelihood of three anomaly files similar to the test set.

Anomaly File 1:

```
> moda1<- setpars(moda1, getpars(fm1))
> fb<-forwardbackward(moda1)
> fb$logLike
[1] -6499.788
```

Anomaly File 2:

```
> moda2<- setpars(moda2, getpars(fm1))
> fb<-forwardbackward(moda2)
> fb$logLike
[1] -37685.79
```

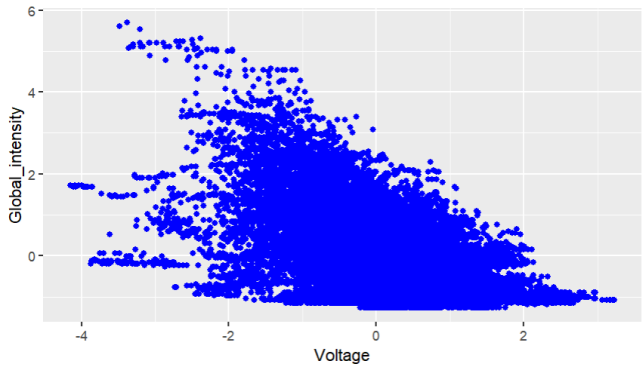
Anomaly File3:

```
> moda3<- setpars(moda3, getpars(fm1))
> fb<-forwardbackward(moda3)
> fb$logLike
[1] -6499.788
```

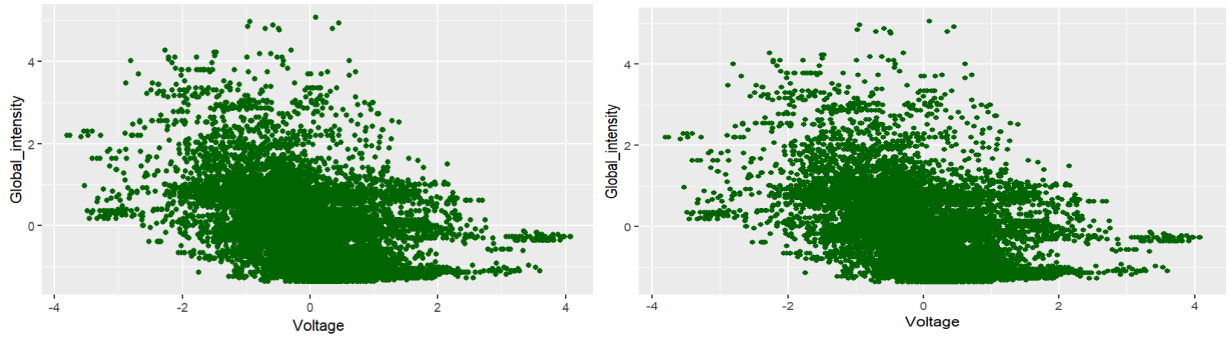
Interestingly, Anomaly File 1 and File 3 have exactly the same log likelihood of -6499.788, which is quite similar to our original model (-5914) and test model (-4488). The 1500 log likelihood difference is relatively small considering the log-likelihood scale of our model, so we cannot conclude that Anomaly File 1 and 3 contain true anomalies. There could be noise in the data. On the other hand, Anomaly File 2 has a significantly different log-likelihood when using our model, indicating that it is highly likely to contain anomalies and requires further investigation.

We are curious as to why Anomalies 1 and 3 are the same and why Anomaly 2 has a large negative log likelihood. Upon further investigation, we discovered that File 1 and File 3 are almost identical. Then, we plot the graphs to see the difference of File 2:

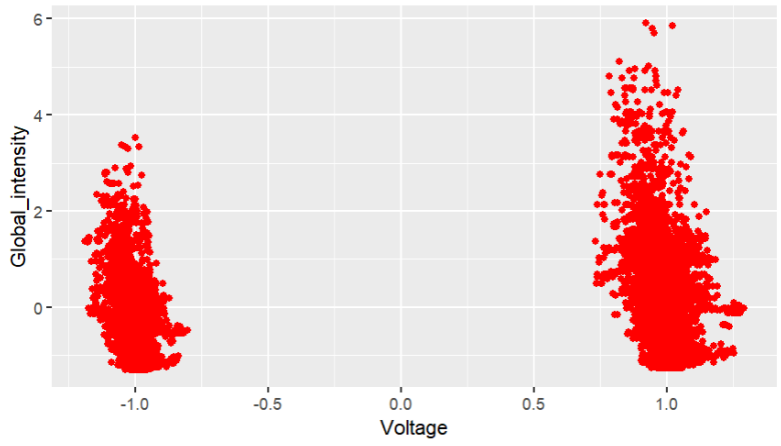
Train set:



File 1 and 3:



File 2:



As we see, although the training data set and files 1 and 3 do have some differences, they follow the same general pattern. However, file 2 is not distributed in the lower middle of the graph, but has two different sections on the sides.

The plots agree with our conclusion above.

We were unable to perform anomaly detection in each file due to our R code reporting errors use library(caret). But here is an outline of what the result should look like this:

The probability density function (PDF) of a multivariate normal is:

$$f(x) = (2\pi)^{\frac{-k}{2}} |\Sigma|^{\frac{-1}{2}} e^{-1/2(x-\mu)'\Sigma^{-1}(x-\mu)}$$ (Berhane).

Then, we center the data: preObj <- preProcess(a1,method="center"), a12 <- predict(preObj,a1)

Calculate variance sigma2: a12= as.matrix(a12), sigma2=diag(var(a12)), sigma2=diag(sigma2)

Probabilities p: A=(2*pi)^(-ncol(X2)/2)*det(sigma2)^(-0.5),

B=exp(-0.5*rowSums((X2%*%ginv(sigma2))*X2)), p=A*B

Now proceed with **Cross-Validation**:

We center the data: preObj <- preProcess(a1$Xval,method="center"), Xval_centered <- predict(preObj,a1$Xval).

Calculate variance sigma2: Xval_centered = as.matrix(Xval_centered),

sigma2=diag(var(Xval_centered)), sigma2 = diag(sigma2)

Calculate pval: A=(2*pi)^(-ncol(Xval_centered)/2)*det(sigma2)^(-0.5)

B = exp(-0.5 *rowSums((Xval_centered%*%ginv(sigma2))*Xval_centered))

pval = A*B

Use pval, determine the best Epsilon.

Finally, find outliers:

X$outliers= X$probability < bestEpsilon

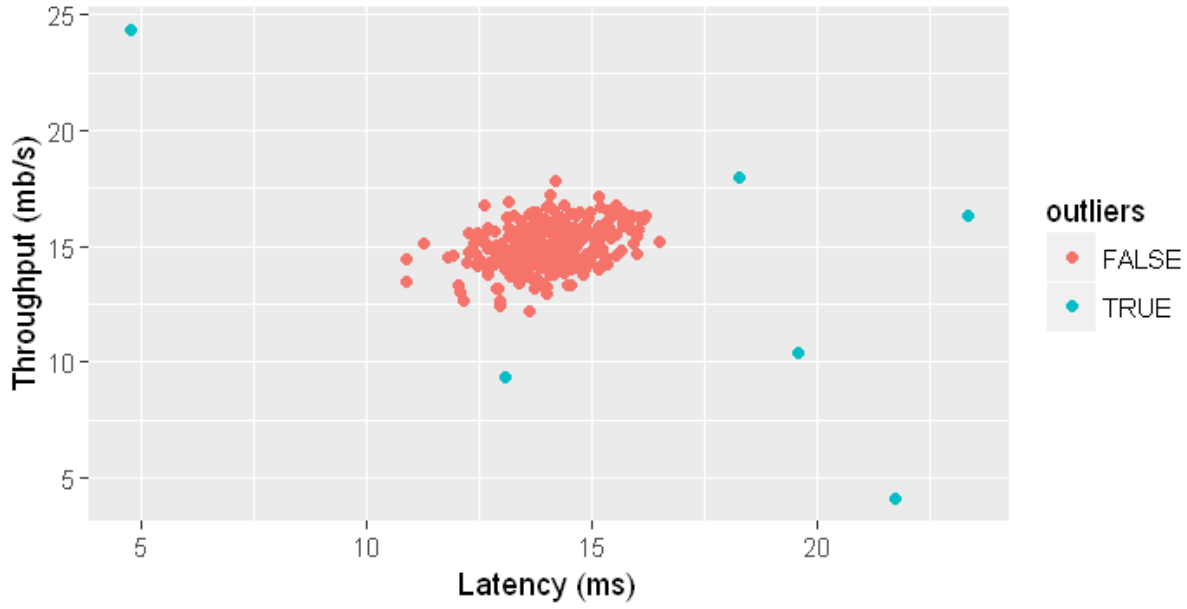
X$outliers=as.factor(X$outliers)

table(X$outliers)

The expected result looks like this:

```
FALSE  TRUE
  301     6
```

The expected visualized result looks like this:



Please note the above Cross-Validation Anomaly Detection part is taken from:

https://datascience-enthusiast.com/R/anomaly_detection_R.html, by Fisseha Berhane.

Not the actual project data. See details on the website.

Challenges

Throughout the project, we encountered several challenges, and we had to make certain assumptions to overcome them:

1. **Missing data values:**

   To handle missing data, we used the na.omit method. However, this method assumes that N/A data do not hold any significant value, which may not be true in practice.

2. **Using only two out of four significant PCA parameters:**

   In our model, we only used voltage and Global_intensity, the top 2 variables suggested by PCA. However, Global_active_power and Sub_metering_3 are also significant variables that we ignored. Neglecting them may lead to less accurate parameter selection.

3. **Time Frame choice:**

   Our time frame selection was based on random dates and not all dates were included. Additionally, we did not define what constitutes a recognizable pattern.

4. **Train model accuracy:**

   Our train model's negative log-likelihood of -5914 may be too large. Although state number 19 is the best value, there are still gaps between log-likelihood and BIC values.

5. **Incomplete Anomaly Detection:**

   While we determined which file contained the most different data points compared to our training model, we still needed to detect each anomaly point in each file.

Lessons learned

In this project, we gained insights into the significance of protecting critical infrastructures, such as electric grids, from anomalies and potential attacks. We also learned about the importance of feature engineering and how PCA can improve model accuracy by identifying the most significant variables. Throughout the HMM training and testing, we encountered various challenges, such as the selection of time frames, proper training of the model, and calculation of log-likelihood using getpars, setpars, and forwardbackward methods. Despite these challenges, we were able to detect anomalies in the test set and discovered that anomaly detection remains a challenging task, and further work is required to identify individual anomaly points in each file. Overall, we learned that a combination of feature engineering and HMM techniques could be useful in identifying potential anomalies in critical infrastructure systems.

Technical part aside, this project also taught us the importance of outlining a group project. Our approach to coding was finishing it part by part, however, we later found out part 3 anomaly detection requires more research than part 1 and part 2. We did not leave enough time for it because we did not plan out the project properly.

Reinforcement Learning Paradigm

Reinforcement learning (RL) is a subfield of artificial intelligence (AI) and machine learning (ML) that focuses on training agents to make decisions in a dynamic environment to achieve a specific goal. It differs from classical machine learning approaches by emphasizing learning through trial and error and feedback from the environment. The given tutorial provides a comprehensive introduction to reinforcement learning and demonstrates how to implement a Q-learning algorithm in Python using OpenAI Gym, a toolkit for developing and comparing reinforcement learning algorithms. Overall, reinforcement learning offers unique advantages in handling complex, real-world problems that involve uncertainty and require decision-making in dynamic environments.

Key Characteristics of Reinforcement Learning

1. Interaction with the environment: In RL, agents learn by interacting with their environment, gathering information, and adjusting their actions accordingly. This trial-and-error process allows agents to discover optimal strategies based on their experiences.
2. State, action, and reward: RL problems are typically modelled using these three components. The state represents the current situation of the agent within the environment, the action is the decision the agent takes, and the reward is the feedback received for a specific action.
3. Goal-oriented learning: Reinforcement learning algorithms aim to optimize long-term cumulative rewards, meaning agents learn strategies that maximize their overall success rather than focusing on immediate gains.
4. Temporal difference learning: RL algorithms often employ temporal difference learning, which uses the difference between estimated future rewards and the current reward to

update the agent's knowledge. This approach allows agents to learn from incomplete sequences of events and accelerates the learning process.

5. Exploration vs. exploitation: A critical aspect of RL is balancing exploration and exploitation. Exploration involves trying new actions to discover their consequences, while exploitation means using the current knowledge to make optimal decisions. By balancing the two, agents can learn to make better decisions over time.

Advantages of Reinforcement Learning over Classical Machine Learning

1. Dynamic decision-making: Reinforcement learning algorithms are designed for decision-making in dynamic environments, making them suitable for real-world applications where conditions change over time. Classical machine learning approaches often require a static dataset and do not adapt as effectively to evolving situations.
2. Learning from limited supervision: Unlike supervised learning, which relies on a large dataset of labelled examples, reinforcement learning agents learn from interacting with the environment and receiving feedback in the form of rewards. This enables learning in situations where obtaining labelled data is challenging or expensive.
3. Continuous adaptation: Reinforcement learning algorithms allow agents to continuously adapt and improve their performance as they gather more experience, making them well-suited for long-term deployments. Classical machine learning approaches, on the other hand, often require retraining when new data is available, making them less adaptable to changing circumstances.
4. Scalability: Reinforcement learning algorithms are highly scalable and can be applied to a wide range of problems, from simple tasks to complex, high-dimensional environments. Additionally, RL can handle continuous action and observation spaces, making it suitable for problems that require fine-grained control.

References:

PCA:

https://www.datacamp.com/tutorial/pca-analysis-r

Standardize data:

https://www.r-bloggers.com/2022/07/how-to-standardize-data-in-r/

Anomaly Detection:

https://datascience-enthusiast.com/R/anomaly_detection_R.html