Examining Anomaly Detection on Power Consumption Using Hidden Markov Model

CMPT 318

Fall 2024

Prof. Uwe Glaesser

Group 15

Kousha Amouzesh, ska387@sfu.ca

Mahdi Beigahmadi, mba188@sfu.ca

Richard Xiong, rxa14@sfu.ca

Matt Friesen, maf7@sfu.ca

Abstract

This project aims to develop a supervised machine learning model for detecting anomalies by analyzing time series data of electric consumption obtained from supervisory control systems. We focus on data preprocessing collected between 2006 to 2009 and will train a Multivariate Hidden Markov Model (HMM) to detect anomalies between our target features. The report evaluates the performance of different models with varying numbers of states, using the Bayesian Information Criterion (BIC) and log-likelihood metrics. The highest-performing model is then utilized in a simulated anomaly detection task; we establish a threshold based on deviations in log-likelihood between the training and validation datasets. We test the model with both real and artificially injected anomalies to demonstrate its effectiveness in identifying significant deviations that may indicate potential threats.

Table of Contents

1. Introduction

The main purpose of this project is to train a model under supervision that can be used to detect threats in cybersecurity. The project is specially designed on machine learning algorithms. The team takes advantage of R to analyze the data in a meaningful and quick way. This project focuses on analyzing a time series data set of electric consumption obtained using supervisory control systems, then designing and training a model to detect anomalies in other data sets. Consequently, we will concentrate on exploring methods for anomaly detection-based intrusion detection utilized for situational awareness analysis of automated control processes. For this project, we will employ Principal Component Analysis to select a suitable subset of variables for training Multivariate Hidden Markov Models. Subsequently, we will utilize these models to assess anomaly detection capabilities.

2. Background

2.1. Technical Background on Data Preprocessing

When it comes to anomaly detection, using the raw data is a massive misstep. The data must be processed first through feature scaling. The most common techniques for feature scaling are normalization and standardization. Standardization, also referred to as Z-Score scaling, scales the data so that the average value is zero, and has a standard deviation of 1 [1]. This is done by taking each instance of the data being scaled and applying the following formula to it: $(X - \mu)/\sigma$ or, in other words, subtracting the mean from the value and dividing by the standard deviation [1]. Normalization, sometimes called min-max scaling, involves adjusting the range of the data to fit between zero and one, inclusively, using the formula $X' = (X - min(X))/(max(X) - min(X))$ [1]. Standardization is best used for data that follows a Gaussian distribution, while standardization is better for data on a consistent scale. However, normalization may compress outliers, making them more difficult to detect [1]. Standardization can make noise and features comparable, while normalization may amplify or

minimize noise depending on the data's distribution. In a Hidden Markov Model (HMM), standardization is best used when multiple features with different ranges and outliers are not extremely disproportionate, which could skew the results. Normalization is best used in an HMM if the data does not follow a normal distribution and when data is collected at regular intervals [1]. Given that the point of the project is to detect anomalies and standardization can still represent outliers in the data, our optimal scaling choice would be standardization.

3. Feature Engineering

Feature selection is a key step in model training. It allows the model to identify patterns between two or more variables and find relationships between them. The best features for model training are those that best describe the behaviour of our dataset and help induce a pattern. In total, we have 7 numeric variables in our dataset, consisting of **Global_active_power**, **Global_reactive_power**, **Voltage**, **Global_intensity**, **Sub_metering_1, Sub_metering_2, and Sub_metering_3**. Due to the overwhelming number of features; we need to break down this higher dimension and choose variables that capture the most important patterns.

3.1. Principal Component Analysis

Principal Component Analysis is a method that reduces the dimensionality of the data. The number of principal components created by maximizing the variance of the data will be much fewer than the number of original variables, thereby reducing the complexity of the data [2]. Practically, we need to estimate the PCA values for all columns or variables. Since our initial dataset size is extensive, we will select a specific day, Mondays, between 9 A.M and 12 P.M, to train our model. However, we will use the entire dataset for the PCA calculation to determine the overall relationships among the variables across all years. It is worth noting that for the time window extraction to take place, we first need to create a new

column containing both the date and time. The next step, involving time window extraction, will be executed as described in Section 4.1 before splitting the data into training and testing sets. The DateTime column conversion is shown below.

```
df$DateTime <- paste(df$Date, df$Time)
df$DateTime <- as.POSIXct(df$DateTime, format="%d/%m/%Y %H:%M:%S", tz = "UTC")
```

In addition, we replaced the NA values in all the columns with `na.approx()`, which interpolates missing values in a numeric vector or column. This is necessary since NA values can further impact the PCA computation, which calculates variance and other important components.

```
fill_na <- function(x) {
  x <- na.approx(x, na.rm = FALSE)# For interpolation
  x <- na.locf(x, na.rm = FALSE)# To handle leading NAs
  x <- na.locf(x, na.rm = FALSE, fromLast = TRUE)# To handle trailing NAs
  return(x)
}
df[numeric_cols] <- lapply(df[numeric_cols], fill_na)
```

The last important thing before carrying out PCA evaluation is standardization. Standardization brings the data points within a mean of 0 and a standard deviation of 1, which is significant for a process like PCA that depends highly on variance comparison between variables. Without standardizing the data, the PCA could be biased if the range of values in one variable is larger than in another. The standardization was done using `df_scaled[numeric_cols] <- scale(df_scaled[numeric_cols])`, where numeric_cols are the collection of numeric columns except time.

Finally, after standardization, we applied the PCA calculation to each numerical variable using the `prcomp()` function and the standardized data we just created.

```
pca_data <- df_scaled[numeric_cols]
pca_result <- prcomp(pca_data, center = FALSE, scale. = FALSE)
```

Here, we computed the variance for each variable and calculated its percentage. Note that the variance is the squared standard deviation.

```
pca_var <- pca_result$sdev^2
pca_var_perc <- pca_var / sum(pca_var) * 100
```

The diagram in Figure 1, shows that PC1 captures the largest portion of the variance in our dataset, accounting for 40.8%. PC1 is the first principal component. PC1 is a linear combination of the original variables that captures the highest variance in the dataset. It represents the direction along which the data exhibits the greatest variation.

```
PC1: 41.81% PC2: 14.73% PC3: 12.47% PC4: 11.54%
 PC5: 9.85% PC6: 8.16% PC7: 1.45% PC8: 0%
```

Figure 1: Percentage of Variance in Multiple PCs

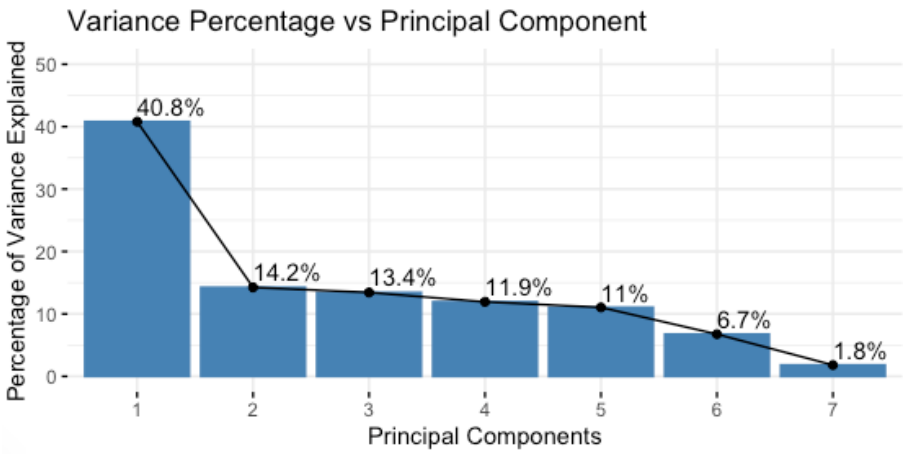


Figure 2: Percentage of Variance in Multiple PCs

By getting a data summary from the computed PCA result using the `prcomp()` function, we can once again see that the standard deviation in Figure 3, which is the square root of the variance, is highest in PC1 and PC2, and the cumulative variance coverage of PC1 and PC2 is ~55% which is enough for our study given that it is more than half of our data and it is easier to visualize.

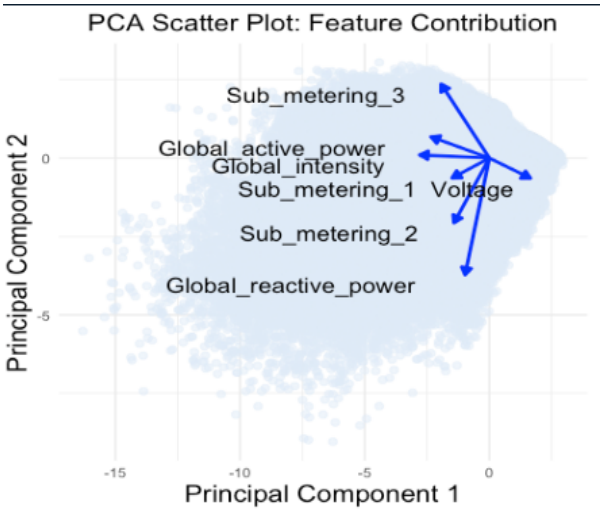
```
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7       PC8
Standard deviation     1.8288 1.0854 0.9987 0.9610 0.88767 0.80791 0.34042 8.111e-15
Proportion of Variance 0.4181 0.1472 0.1247 0.1154 0.09849 0.08159 0.01449 0.000e+00
Cumulative Proportion  0.4181 0.5653 0.6900 0.8054 0.90392 0.98551 1.00000 1.000e+00
```

Figure 3: The Importance of The Components and Their Values

3.2. Feature Selection

To gain a deeper understanding of the impact of each feature on our data variance, we can visualize a PC1 vs PC2 plot and show the correlation of each feature with the two principal components with vectors in Figure 4. More clearly, when PCA results are processed, a matrix called the loading matrix is calculated. Each entry on this matrix indicates the contribution or weight of the feature on the principal components PC1 and PC2. These weights are then used as the slope of the arrow in the form (PC1, PC2) coordinate.

Figure 4: Feature Contribution on PC1 and PC2



According to Figure 4, variables like **Sub_metering 1, 2, 3** and **Global_active_power**, **Global_reactive_power** and **Voltage** have the most correlation influence on both PC1 and PC2 as their slope is heavily influenced by both the magnitude of their PC1 and PC2 values. In contrast, variables like **Global_intensity** are strictly correlated with PC1 and have minimal influence on PC2. To execute our feature selection with higher confidence, we can also look

at the variables contributing at the highest magnitude to the principal component 1, which is the most important principal component. We can obtain the variable with the maximum PC magnitudes from Figure 5.

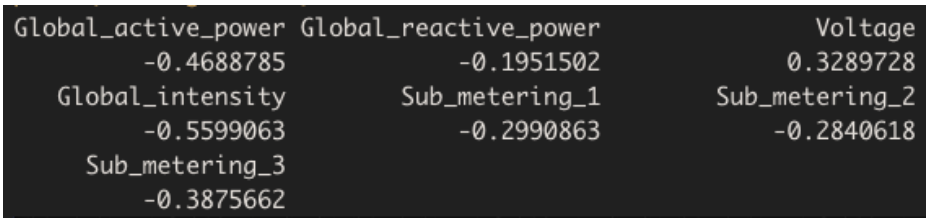


Figure 5: The PC Values of all the Power Factors

Given that **Global_active_power**, **Global_intesity** and **Voltage** give the highest PC magnitudes, our choice would be **Global_intensity** and **Voltage** as our two features. We could have also chosen **Global_active_power** as the second variable. However, due to poor convergence during training, we decided to go with **Voltage** instead. Also, note that the **Global_intensity** has the highest magnitude among the other variables.

4. Hidden Markov Model

The Markov model explains the probability of a series of random variables and states that each can take on values from some set [3]. "The term "hidden" is used because the states themselves are not directly observable; instead, only the outputs (or observations) generated by these states are visible [4]. This is a key characteristic that distinguishes HMMs apart from Markov chains" [6]. This anomaly detection method employs Hidden Markov Models (HMMs), which are robust statistical tools for modelling generative sequences characterized by an underlying process that generates an observable sequence. The nature of the project, which is for anomaly detection, is highly compatible with such models. Therefore, the anomaly detection method can train optimal Hidden Markov Models to determine the normality of each entry in the observed data and subsequently locate anomalies by identifying any deviations from this calculated normality.

4.1. Testing and Training Data Division

Prior to doing the train and test split, it is necessary to take a subset of our data for Mondays from 9 A.M to 12 P.M. This allows the model to focus on the same time of the day for all the 3 years and find more meaningful patterns. The 9 A.M to 12 P.M timeframe was chosen because it captures the typical increase in power consumption as people begin their workday, a period known to be one of the most active hours. To extract this timeframe, a function is implemented called `extract_time_window.` To ensure that the trained model has an acceptable level of accuracy with the lowest chance of overfitting, we have dedicated years from 2006 to 2008 to the training set and 2009 to the test set. This distribution yields 70% of our data to the training and 30% to testing, which is a reasonable approach since 30% for test data allows us to ensure there is minimal to no overfitting in our model. Also, with the majority of our data being for training, the model can learn the pattern from a wider variety of observations to succeed during testing.

```
extract_time_window <- function(dataframe) {
  df_monday_9am_to_12pm <- dataframe %>%
    filter(weekdays(DateTime) == "Monday" &
             hour(DateTime) >= 9 & hour(DateTime) < 12)
  return(df_monday_9am_to_12pm)
}
df_scaled <- extract_time_window(df_scaled)
train_data <- df_scaled %>% filter(Year <= 2008)
test_data <- df_scaled %>% filter(Year == 2009)
train_features <- train_data[, c("Global_intensity","Voltage")]
test_features <- test_data[, c("Global_intensity","Voltage")]
```

4.2. Model Optimizations Methods

We developed a model that faced a significant computational challenge during training. As the number of states increased, the model's computational speed progressively declined. To address this issue, we opted to limit the number of states to a maximum of **13**. This optimization significantly enhanced the code's performance, but we anticipated a faster computational speed. Consequently, we reduced the number of seeds within the

"set_seed()" function. This function guarantees the precise reproducibility of all calculations [5]. With fewer seeds, the training process became substantially faster compared to the previous iterations.

4.3. Training Functions and Best Model Selection

This model trains utilizing the "depmixS4" package. It enables the training of HMMs with varying numbers of states and accelerates the training process by concurrently utilizing multiple processor cores. Also, the training process commences by initializing potential states and configuring the expectation maximization (EM) algorithm parameters through the "em.control" function. Subsequently, lists are prepared to store relevant information, such as log-likelihoods and BIC values. To enable fast parallel processing, a backend was established using the "doParallel" and "foreach" packages. The backend was configured to utilize all available processor cores except one with the "detectCores" and "makeCluster" functions. Each model's training is executed in parallel. The "depmix" function defines the HMM, while the "fit" function performs the optimization silently. Metrics, including log-likelihood and BIC, were computed for each model configuration. After the computation, the "stopCluster" function terminates the parallel setup. The resulting data is subsequently combined, presenting the log-likelihood and BIC values for each model configuration. Here is the code that trains the model:

```
states_list <- c(4, 6, 7, 8, 10, 12, 13)
em_ctrl <- em.control(maxit = 1000, tol = 1e-5)
num_cores <- detectCores()
cl <- makeCluster(num_cores)
registerDoParallel(cl)
results <- foreach(num_states = states_list, .packages = 'depmixS4') %dopar% {
  suppressMessages({
    hmm_model <- depmix(
      response = list( Global_intensity ~ 1, Voltage ~ 1),
      data = train_features,
      nstates = num_states,
      family = list(gaussian(), gaussian())
    )
    set.seed(42)
    print(paste0("train model state = ", num_states))
    fitted_model <- fit(hmm_model, ntimes = 10, verbose = FALSE, emcontrol = em_ctrl)
```

```
    log_likelihood <- logLik(fitted_model)
    bic_value <- BIC(fitted_model)
    list(
      num_states = num_states,
      log_likelihood = log_likelihood,
      bic_value = bic_value,
      model = fitted_model
    )
  })
}
```

After running the program, the model gave us the following results in figure 6:

```
Log-Likelihood for 4 states: -13489.81
BIC for 4 states: 27285.46
Log-Likelihood for 6 states: -8933.939
BIC for 6 states: 18449.96
Log-Likelihood for 7 states: -6146.814
BIC for 7 states: 13043.43
Log-Likelihood for 8 states: -4774.524
BIC for 8 states: 10486.3
Log-Likelihood for 10 states: -1177.786
BIC for 10 states: 3726.917
Log-Likelihood for 12 states: 2419.198
BIC for 12 states: -2954.031
Log-Likelihood for 13 states: 1902.969
BIC for 13 states: -1635.466
```

Figure 6: BIC and Log-Likelihood for all the states

Based on the results from Figure 6, we can see a drop in BIC values and an increase in log-likelihood values with the increment of the state numbers up to 13 states. This trend demonstrates an ideal behaviour and confirms our choices of features for HMM training, as a higher number of states increases the model's complexity, enabling a better fit to the training data. However, we will determine the optimal number of states after evaluating the model's performance on test data.

4.4 Bayesian Information Criterion and Log-likelihood Interpretation

The term BIC is a method often used in statistical computation due to its simplicity and effectiveness [6]. The purpose of using this formula is to minimize the odds of overfitting. The lower the BIC means, the less chance of overfitting for the model. The formula that is used for calculation is $k * ln(n) - 2 * ln(L)$, where k is the number of parameters the

model considers, n is the number of observations or sample size, and L is the probability function p(x | θ, M), with M being the model itself, x being the observed data, and θ being the parameter values that maximize L[7]. On the other hand, log-likelihood determines how well a statistical model fits the observed data. The formula for this concept is $Log\ L\ (\theta)$ and $L(\theta) = P\ (data \mid \theta)$ is the formula for the log-likelihood function, where θ is the parameter of the model [8]. We compared BIC and log-likelihood from multiple points of view and how they are related to the model and states in our program. Figure 7 completely describes the behaviour of BIC and log-likelihood in the model we trained.

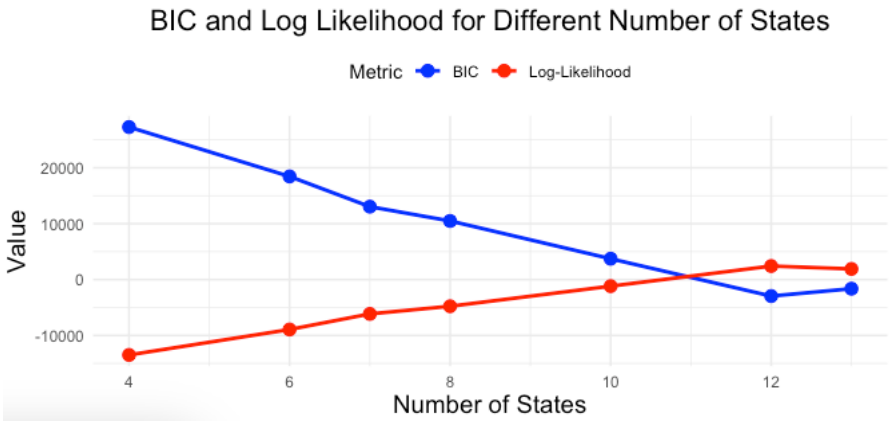


Figure 7: BIC and Log-Likelihood Plot

Figure 7 presents the values on the y-axis and the number of states on the x-axis. The log-likelihood starts with four states and an initial value of less than -10,000. Based on Figure 7, as the number of states increases, the log-likelihood value increases consistently. The graph highlights a peak in the log-likelihood around 3,000, corresponding to values in the y-axis. On the other hand, the BIC begins at state 4 with a high value around 30,000 and then decreases significantly as it progresses to subsequent states. The minimum value for BIC occurs at state 12 and -3,000.

Even though the model with 7 states does not have the highest log-likelihood, it has a significantly lower BIC than models with fewer states, indicating a better balance between

model complexity and fit. We compared cases where different numbers of states were chosen as the best and tested the models on the test data. As shown in the table below, the difference between the train and test log-likelihoods for the 7-state model is the least, which is why we have selected it as the optimal model.

Table 1: Train and Test Log-Likelihood Comparison

Number of States (cases)	Train Log Likelihood	BIC	Test Log Likelihood
6	-8933.939	18449.960	-8328.112
7	-6146.814	13043.427	-5609.337
8	-4774.524	10486.297	-6679.709
10	-1177.7786	3726.917	-5728.307
12	2419.198	-2954.031	-4006.831
13	1902.969	-1635.466	-1967.591

For further evidence, Figure 6, which illustrates the training loop, shows a significant improvement in both log-likelihood and BIC when transitioning from 6 to 7 states, suggesting that adding another state provides a meaningful improvement in model fit without excessive complexity. While continuing to improve log-likelihood, the move from 7 to 8 states offers a smaller reduction in BIC, indicating diminishing returns in terms of balancing fit and complexity.

4.5 Model Evaluation and Prediction

Model evaluation and testing are crucial steps in model training. We want to assess how well our model performs on data it has not been tested with before. To achieve this, We set aside 30% of the data for testing. Below, We will explain the steps we took to develop an algorithm for evaluation. The code block below and Figure 10 demonstrate the process, and the output shows a log-likelihood of "**-5609.337"** or Normalized **"-0.6492288 "** for the test data compared to the original and the normalized versions **"-6146.814", "-0.3191492"** on the train data. This indicates that our trained model has a similar level of effectiveness in understanding and explaining the observed data (training data) and the unobserved data (test

data). Also, it is worth mentioning that there is a slight reduction in the likelihood of the test data being natural due to unexpected patterns, which could be anomalies.

```
test_model <- depmix(
  response = list(Global_intensity ~ 1, Voltage ~ 1),
  data = test_features,
  nstates = best_num_states,
  family = list(gaussian(), gaussian())
)
test_fitted <- setpars(test_model, getpars(best_model))
fb_test <- forwardbackward(test_fitted)
test_log_likelihood <- fb_test$logLike
cat("Log-Likelihood on Test Data:", test_log_likelihood, "\n")
```

```
> cat("Log-Likelihood on Test Data:", test_log_likelihood, "\n")
Log-Likelihood on Test Data: -5609.337
```

Figure 8: Log-likelihood Result on Test Data

5. Anomaly Detection

Anomaly detection uses a training and validation set to determine a threshold for detecting anomalies in unseen data. An anomaly is detected when the standard deviation of the given data exceeds the threshold. Upon detection, a flag is raised in the system. From the training and validation set, we obtained the threshold. We determined the threshold to be **1.580539**. The training set contains data from 9 A.M to 12 P.M every Monday from 2006-2008. The validation set contains data from 9 A.M to 12 P.M every Monday from 2009, partitioned into 10 equal subsets for determining a threshold. We tested each partitioned set with the training model and calculated their log-likelihood. The log-likelihood of the training and validation set is then standardized, and the difference between the two values is recorded in Table 2. The value with the greatest difference from the training data's log-likelihood is set as the threshold.

Partition # of test set	Log-Likelihood	Normalized Log-Likelihood	Deviation
1	142.25891	0.1646515	0.483800719
2	-277.44931	-0.3211219	-0.001972692
3	145.02019	0.1678474	0.486996641
4	-1281.21899	-1.4828924	-1.163743157
5	-1271.95630	-1.4721716	-1.153022451
6	-320.33589	-0.3707591	-0.051609933
7	-1641.33021	-1.8996877	-1.580538542
8	-482.13506	-0.5580267	-0.238877498
9	-90.38115	-0.1046078	0.214541386
10	-591.60922	-0.6847329	-0.365583695

Table 2: Log-likelihood, Normalized Log-likelihood; and Deviation of the 10 Test Data Partition

5.1. Log-Likelihood of Training and Testing Data

We calculated the log-likelihood of the training and test data to be **-6146.814** and **-5609.337**, respectively. The normalized training and test data values are **-0.3191492** and **-0.6492288,** as shown in Figure 8**.** Table 2 provides a visual comparison of the differences between the normalized log-likelihoods.

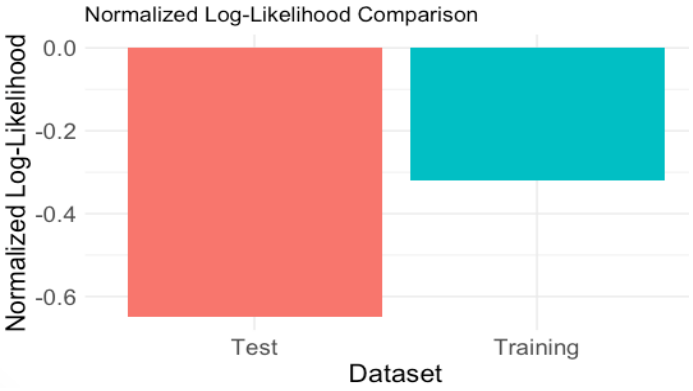


Figure 8: Normalized Log-Likelihood Comparison: Training vs Test

5.2. Testing with Injected Anomalies

We tested the model by partitioning the test set into three subsets and modifying them to simulate anomalies from real data. The training model detected anomalies in the subset, with

each subset deviating from the threshold by **4.90229, 9.996295, and 6.212019,** respectively. We further tested with the unmodified data, and the results obtained were no anomalies detected in the first and third subsets and anomalies detected in the second subset, which exceeds the threshold by **4.838944.** We assumed the anomalies detected in the unmodified subset 2 came from the original subset containing anomalies. Figure 9 shows the anomalies spread from the unmodified and modified subsets 1 and 2 of the two features we chose.

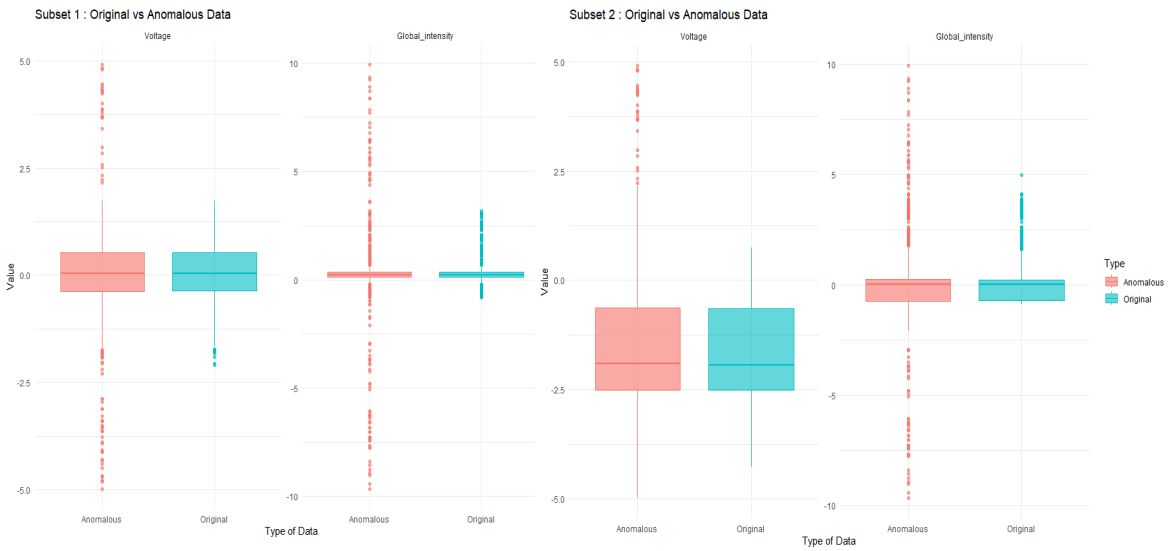


Figure 9: Spread of Anomalies From the Unmodified and Modified Subset

6. Conclusion

6.1. Challenges and Lessons

As we moved forward with our experiment, we faced several challenges, such as feature selection using PCA. Initially, we selected features with the highest loading magnitudes for principal components 1 and 2, specifically **Global_intensity** and **Global_active_power.** However, this caused the BIC and log-likelihood values to fail to converge close enough to zero. Changing to **Global_intensity** and **Voltage** resolved this major problem, enabling smooth convergence and improved model performance on both training and testing datasets. This highlighted the need for PCA and model experimentation for optimal feature selection.

Another challenge was determining the ideal number of states for the HMM. While models with more than 7 states achieved higher log-likelihood and lower BIC measures, they performed poorly on test data due to overfitting on the test data [9]. The 7-state model, in contrast, offered more stable results, balancing complexity and accuracy. Measuring log-likelihood on both training and testing data proved to be a significant step for ensuring the model's performance to unseen sequences.

6.2. Project Results

Finally, despite the listed challenges, our model demonstrated a strong capability to identify the injected anomalies using the 7-state HMM. It successfully captured the relationship between Voltage and Global Intensity, our variables of interest, while achieving a well-balanced trade-off between log-likelihood and BIC values across both training and testing data. The procedures applied in this project effectively balanced the accuracy and complexity of the HMM, generating a positive outcome and reinforcing the ability for anomaly detection tasks.

7. References

[1] R. Alam, "Normalization vs. Standardization Explained," *Medium*. [Online]. Available: https://towardsdatascience.com/normalization-vs-standardization-explained-209e84d0f81e. [Accessed: Nov. 20, 2024].

[2] Q. Guo, W. Wu, D. Massart, C. Boucon, and S. De Jong, "Feature selection in principal component analysis of analytical data," *Chemometrics and Intelligent Laboratory Systems*, vol. 61, no. 1–2, pp. 123–132, Feb. 2002, doi: 10.1016/s0169-7439(01)00203-9.

[3] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

[4] O. Pramod, "Hidden Markov models: The secret sauce in natural language processing," *Medium*. [Online]. Available: https://medium.com/@ompramod9921/hidden-markov-models-the-secret-sauce-in-natural-language-processing-e05892d52963#:~:text=The%20term%20%E2%80%9Chidden%E2%80%9D%20is%20used,%E2%80%9D%2C%20let's%20consider%20an%20example. [Accessed: Nov. 19, 2024].

[5] DatAnnihilyst, "Set seed function," *Posit Community*. [Online]. Available: https://forum.posit.co/t/set-seed-function/89599. [Accessed: Nov. 19, 2024].

[6] A. A. Neath and J. E. Cavanaugh, "The Bayesian information criterion: background, derivation, and applications," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 2, pp. 199–203, Dec. 2011, doi: 10.1002/wics.199.

[7] E. Wit, E. van den Heuvel, and J.-W. Romeyn, "'All models are wrong...': an introduction to model uncertainty," *Statistica Neerlandica*, vol. 66, no. 3, pp. 217–236, Sep. 2012, doi: 10.1111/j.1467-9574.2012.00530.x.

[8] Alexander, "Log likelihood function," *Statistics How To*. [Online]. Available: https://www.statisticshowto.com/log-likelihood-function/. [Accessed: Nov. 20, 2024].

[9] E. Aarts, "Multilevel HMM tutorial," *cran.r-project*. [Online]. Available: https://cran.r-project.org/web/packages/mHMMbayes/vignettes/tutorial-mhmm.html. [Accessed: Nov. 20, 2024].

8. Contributions

1. **Kousha Amouzesh**, ska387@sfu.ca **(contribution : 35%)**

   Programming Contribution:

   - Code for PCA and data cleaning with Mahdi
   - Code for Feature Selection with Mahdi
   - Code for the Training Scripts and Log-likelihood and BIC plots

   Report Writing Contribution:

   - Collaborated in writing the Abstract and Table of Content
   - Wrote the reports for sections 3.1, 3.2, 3.3, 6.1, and 6.2
   - Formatted the paper and proofread the text

2. **Mahdi Beigahmadi**, mba188@sfu.ca **(contribution: 28%)**

   Programming Contribution:

   - Code for Feature Engineering with Kousha
   - Code for Feature Scaling (Standardization)
   - Code for PCA with Kousha

   Report Writing Contribution:

   - Wrote report for section 4 including 4.1, 4.2, 4.3, 4.4, and 4.5
   - Formatted and adjusted the reference page
   - Helped in writing abstract and summary sections
   - Text revision

3. **Richard Xiong**, rxa14@sfu.ca **(contribution: 25%)**

Programming Contribution:

- Code for Anomaly detection
- Code for injecting Anomalies

Report Writing Contribution:

- Wrote section 5 of Anomaly detection
- Formatted the paper and proofread the text

4. **Matt Friesen**, maf7@sfu.ca **(contribution: 12%)**

   Report Writing Contribution:

   - Wrote introduction
   - Wrote part of section 4.4 with Mahdi
   - Various edits and revisions on multiple sections