A Novel Framework for Feature Selection Using Automaton and Hidden Markov Models

Mohsin Ali, Jitendra Choudhary, Anshul Shrotriya, Anil Patidar

***Abstract*—Feature selection is a crucial pre-processing step that aims to improve the computational efficiency and performance of models by identifying the most relevant features. In this study, we propose a novel framework for feature selection that leverages automata and hidden Markov models (HMM). This framework employs probabilistic modeling and state transitions to rank features based on their significance while maintaining computational efficiency. With five different datasets, we compare the proposed framework to RFE, L1 Regularization, mutual information, recursive feature elimination (RFE), Boruta, and Random Forest. Additionally, we evaluate the Automaton HMM framework based on classification accuracy, RMSE, MSE, R^2, memory usage, and time complexity. The results indicate that for both large and small datasets, the accuracy metrics of the proposed framework outperform all other methods. Furthermore, memory and time usage are lower than RFE, Lasso (L1), and mutual information on larger datasets. The Friedman test confirms statistically significant improvements over all other approaches. In particular, HMM-based feature selection cuts down on computational memory and time while improving classification accuracy. This is shown by average improvements in accuracy and big drops in feature dimensionality that don't hurt model performance. This framework offers a robust and scalable solution for feature selection, especially in domains that require rapid and accurate decision-making, such as the Internet of Things, time series systems, and high-dimensional machine learning applications.**

***Index Terms*—Automaton, HMM, Feature Selection, RFE, Lasso, MI, Friedman Test.**

I. Introduction

THE rapid development of data makes feature selection a core component in machine learning, big data, and statistics. High-dimensional datasets are commonly encountered in fields like finance, text classification, biology, medicine, genomics, and chemistry [1-6].

Feature selection [7-10] is a technique in machine learning and data analysis that is used to select the important feature for training any model. This technique improves any model performance by reducing the dimensionality and fitting, as well as improving the computational time and use in a wide range of applications [11, 12]. Today, many feature selection techniques are available, including the filter method [13], wrapper method, and embedded method. The filter method does not depend on the particular machine learning algorithm or its features found via statistical techniques like correlation and mutual information. The filter approach usually misses the information about classification, which causes a decrease in accuracy during the learning process.

In the wrapper method [14, 15], the feature is selected based on the previous phase to determine whether to add or remove it from the chosen feature set. The wrapper method is computationally expensive in terms of cost and time, which is high compared to the filter method. The wrapper method includes recursive feature elimination (RFE), forward selection, and backward elimination.

The third category of feature selection is the embedded method; embedded methods integrate feature selection directly into the training process of a machine learning algorithm. Unlike the filters and wrappers method, these methods use the internal mechanisms of models to evaluate the relevance of features. Moreover, high-dimensional applications such as genomics, finance, and NLP also employ the embedded method.

The Mutual Information (MI) [38] is another algorithm that measures the statistical dependence between two random variables. The MI measures how much knowing the value of one variable reduces the uncertainty of the other variable. Additionally, MI also accounts for non-linear relationships, consistency, and changes in random variables using the Pearson correlation coefficient.

Another method is Lasso L1 regularization [39], which is used to select the feature selection and also minimize the absolute mean square error of the coefficient. As a result, it improves the predictions and accuracy of the model by combining the ridge regression and subset selection.

Among the conventional methods, recursive feature elimination is a popular technique, particularly used for supervised learning tasks. It systemically removes the least important feature at each step, retains the model, and, at every iteration, identifies a subset of features that contribute the most to the model's performance. RFE is effective in many applications [16-24], but its limitation is that it is computationally expensive. Additionally, in the case of MI and Lasso (L1) regularization, both algorithms take a large amount of memory.

Another critical limitation of the convolutional approach to feature selection is the capture of latent structures or dependencies in the data. This gap is problematic for sequential data with intricate interdependencies. Furthermore, hidden feature relationships could be more easily quantifiable using linear or statistical techniques.

To remove this limitation, we propose a novel framework to identify the feature selection, which decreases the time and memory usage, especially on large datasets, and enhances the accuracy. The novel approach combines two concepts, Hidden Markov Models (HMM) [25-29] and automatons. Hidden Markov models are statistical tools that describe

Manuscript received February 18, 2025; revised July 5, 2025.

Mohsin Ali is an Assistant Professor of Computer Applications, Medicaps University, Indore 453331, INDIA (e-mail:coolbuddy.next.door@gmail.com).

Jitendra Choudhary is an Associate Professor of Computer Science, Medicaps University, Indore 453331, INDIA (e-mail:jitendra.scsit@gmail.com).

Anshul Shrotriya is an Assistant Professor of Department of Electronic Engineering, Medicaps University, Indore 453331, INDIA (e-mail:anshul.shrotriya@medicaps.ac.in).

Anil Patidar is an Assistant Professor in Department of Computer Applications, Medicaps University, Indore 453331, INDIA (e-mail:anilpatidar2404@gmail.com).

how a system changes over time based on hidden factors, making them ideal for situations where events are connected in a sequence, like time. By estimating the likelihood of observing features with a given specific hidden state, HMM provides the probabilistic measure to select the feature.

This approach employs HMM to calculate log-likelihood scores for each feature, reflecting their importance in representing the dataset's latent patterns. Furthermore, these calculated scores are then fed into an automaton, which applies predefined transition rules and thresholds to categorize features as "useful" or "not useful." This approach based on automata gives a stronger and more organized way to choose the feature, making up for the flaws of other methods and working with datasets that have hidden or non-linear dependencies.

In addition, the proposed approach works on large and small datasets. For larger datasets (greater than 2000), we set the hidden state to 2, and for small datasets, the hidden state will be 8. Furthermore, to evaluate the proposed method, we used 5 different datasets, which are widely used in the respective fields: RT-IoT22, the flight dataset, which belongs to the time series, the adult dataset, diabetes, and finally the wine datasets.

This paper is organized into several sections. Section 2 provides an overview of the background work, while Section 3 details the proposed Hidden Markov Model (HMM) and automaton. Section 4 emphasizes the experimental analysis, and Section 5 discusses the results of the proposed method. Finally, Section 6 concludes the paper.

II. Background Work

This section provides background work information on HMM, RFE, and automaton for feature selection.

In 2002, Isabelle Guyon [30] et al. proposed a method for finding feature selection. The author specifically focused on finding gene selection using Support Vector Machine methods based on Recursive Feature Elimination (RFE). The author uses two datasets, which are the leukemia and colon cancer datasets. The RFE method in both datasets achieves more accuracy than the baseline method.

Le-Bing Zhang et al. [31] introduce a method to detect fake faces using advanced texture and color analysis with a combination of machine learning. The author also increases the system's speed with the help of SVM-RFE, which picks only the most important features and removes unnecessary ones.

Hyelynn Jeon and Sejong Oh [32], authors, proposed an ensemble approach for improving the RFE called Hybrid RFE, which is the combination of SVM-RFE, RF-RFE, and GBM-RFE and combines their features via weighting functions—the weight functions of type simple sum and weighted sum. Additionally, the author evaluates the ensemble approach using eight different datasets.

Dheeb Albashish [33], the author, presents a hybrid metaheuristic model designed to solve the feature selection problem based on binary biogeography optimization with SVM RFE. The author used eight different datasets to evaluate the model and also compare it with another filter method in terms of accuracy and number of selected features. The model results indicate that the BBO-SVM-RFE is reliable for searching the feature space to obtain optimal feature combinations.

Mohammed Awad and Mohammed Awad [34] propose a method called RFE with Cross-Validation using a Decision Tree. This method selects the optimal subset of the cluster, which is 15 from the 42 features of UNSW-NB15, and evaluates it using several ML classifiers, like decision trees and tree-based ones. The proposed method achieves 95.56% compared to the traditional approach of 95.30%.

B. Richhariya [35] proposed a new technique called Universum Support Vector Machine-based Recursive Feature Elimination (USVM-RFE). Unlike SVM-RFE, which focuses only on local data, USVM-RFE incorporates broader and global information about the data during the feature selection process. The author used this method and evaluated MRI data from Alzheimer's disease, analyzing different brain tissue like grey matter and white matter. The result indicates that the new technique has achieved better accuracy than SVM-RFE in classifying control normal, mild cognitive impairment, and Alzheimer's disease patterns.

Li Zhang [36] proposed the Gaussian kernel support vector machine recursive feature elimination method, which ranks the feature in a nonlinear way. This paper addresses the computational challenges of GKSVM-RFE and introduces two faster versions called Fast GKSVM-RFE (FGKSVM-RFE). These methods aim to make the feature elimination process quicker. Furthermore, the author uses first-order and second-order approximation schemes with approximate Gaussian kernels.

Stephen Adam et al. [37] propose a model for finding feature selection from the Hidden Markov and Semi-Markov models. This paper introduces a new parameter called feature saliencies, which distinguishes between the states. The feature saliencies represent the probability relevant to the distinction between state-dependent and state-independent. We use an EM algorithm to calculate the MAP estimate for model parameters. Furthermore, the prior probability calculates the model cost and feature selection.

While Recursive Feature Elimination (RFE), MI, Boruta, Lasso, and Hidden Markov Models (HMM) help understand which features are important, they do not have a clear way to confidently choose the final set of features, and none of the authors address how to save time and memory.

To address this gap, we propose integrating an automaton-based framework, which applies logical transition and rules based on log-likelihood thresholds. This new combination enhances the ability of HMM to identify hidden patterns and the logical consistency of automaton, allowing for a more effective and reliable way to select features.

III. Proposed Methodology

In this section, we elaborate on each section of the automaton, from data pre-processing to final feature selection.

A. Automaton Transition Diagram

The automaton flow diagram Fig. 1 encapsulates the feature selection process combining automaton theory and Hidden Markov Models (HMM). It consists of sequential transitions (S1 to S6, dead state), representing distinct feature evaluation steps. We will elaborate each step based on the

performed mathematical function, ensuring that each automaton step aligns with both theoretical and practical aspects. An automaton $\mathcal{S}=(Q,\Sigma,\delta,q_0,F)$ is constructed, where:

1) Q : A finite set of states representing potential decisions for each feature ($q_{Useful}, q_{\text{NotUseful}}$).
2) Σ : The alphabet consisting of inputs such as log-likelihood scores $\mathcal{L}_i$ of each feature.
3) δ:$Q\times\Sigma\rightarrow Q$: The transition function based on the comparison between $\mathcal{L}_i$ and the threshold τ.
4) q_0 : The initial state ($q_{\text{Undecided}}$).
5) F : A final states (q_{Useful} , $q_{\text{NotUseful}}$).

1) S1: Represent the input State: In S1, this state serves as the starting point where we input data using the sklearn library and read the data file with the pandas library. Given a dataset X with m samples and n features:

$$X=\{x_{ij}|i\in[1,m],j\in[1,n]\}$$

And $y=$ Target.

2) S2: Data Discretize State: From the S1 state, it takes the input, and after processing it, it goes to the S2, which indicates that the data discretization process is complete. If S2 fails, it goes to a dead state, indicating a data failure has occurred. The role of the S2 state is mathematically expressed as follows: Each column x_j of X is discretized into q quantile bins:

$$D_{ij}=Q\left(x_{ij},q\right)$$

where $Q(x,q)$ is the quantile function that maps a continuous value x into one of q discrete bins. Each value in x_j is assigned a label l in the range $\{0,1,\ldots,q-1\}$.

3) S3: One hot encoding: In State S3, the S2 state gets the input, converts it into one hot encoding, and reaches the state S3 after completing the process. After reaching S3, we still change the parameters of one hot encoding and re-run the method. The State S3 is mathematically represented by: For each discretized feature D_j, one-hot encoding transforms each bin label into a binary vector representation:

$$E_{ij}=[e_1,e_2,\ldots,e_q]$$

where $e_k=1$ if x_{ij} belongs to bin k_{r} and $e_k=0$ otherwise.Thus, the transformed datasets E becomes:
$E=\{E_j|j\in[1,n]\}$, with E_j as the one-hot-encoded matrix of column x_j.

4) S4: Hidden Markov Model and Log Likelihood calculation: The automaton transitions from state S3 to S4 upon receiving the log-likelihood scores as input. After completing the transition for a column, the process iteratively evaluates the remaining columns. In the S4 state, each feature is modeled using a Hidden Markov Model (HMM) We model each column E_j using a Hidden Markov Model (HMM) with k hidden states.

$$\theta_j=(\pi,A,B)$$

where:

1) π is the initial state distribution.
2) A is the state transition probability matrix:

$$A=\{a_{kl}\},\ a_{kl}=P\left(z_{t+1}=l|z_t=k\right)$$

3) B is the emission probability matrix:

$$B=\{b_{kv}\},\ b_{kv}=P\left(x_t=v|z_t=k\right)$$

For the sequence of observations $E_j=\{e_1,e_2,\ldots,e_m\}$, the likelihood is given by:

$$P\left(E_j|\theta_j\right)=\sum_{z_1,z_2,\ldots,z_m}\pi_{z_1}b_{z_1e_1}\prod_{t=2}^{m}a_{z_{t-1}z_t}b_{z_te_t}$$

Furthermore the EM algorithm is used to estimate following parameter where in E-step we Compute the expected state probabilities and in M-step we update the parameters. E-Step:

$$\theta_j=(\pi,A,B)$$

$$\gamma_t(k)=P\left(z_t=k|E_j,\theta_j\right)$$
$$\xi_t(k,l)=P\left(z_t=k,z_{t+1}=l|E_j,\theta_j\right)$$

M-step: Update π, A, and B :

$$\pi_k=\gamma_1(k)$$
$$a_{kl}=\frac{\sum_{t=1}^{m-1}\xi_t(k,l)}{\sum_{t=1}^{m-1}\gamma_t(k)}$$
$$b_{kv}=\frac{\sum_{t=1}^{m}\gamma_t(k)\cdot\mathbb{I}(e_t=v)}{\sum_{t=1}^{m}\gamma_t(k)}$$

After iterating each column via the EM step, then for each feature x_j, the log-likelihood of the observed sequence E_j under the fitted HMM θ_j is

$$L_j=\log P\left(E_j|\theta_j\right)$$

. This score measures how well the HMM explains the observed data for feature x_j. If it fails to perform the HMM operation, the process will stop and enter the dead state.

5) State 5: Threshold State: The S4 state takes the input as the loglikelihood score for each feature and reaches the S5 state using the 25th percentile threshold function, denoted by τ. In Fig. 1, the transition state from S4 to S5 signifies the adjustment of the threshold value.

$$\tau=\text{Percentile}_{25}\left(\{\mathcal{L}_1,\mathcal{L}_2,\ldots,\mathcal{L}_n\}\right)$$

For each feature i, the log-likelihood score $\mathcal{L}_i$ is compared to τ :

$$\delta\left(q_0,\mathcal{L}_i\right)=\begin{cases} q_{\text{Useful}}\ , & \text{if } \mathcal{L}_i>\tau \\ q_{\text{NotUseful}}\ , & \text{if } \mathcal{L}_i\leq\tau \end{cases}$$

6) State 6: Final State: In the final state, S6, the output features greater than the threshold are in the final state from S5. For the "Useful" feature (q_{Useful}), these indices are included in the set of selected features:

$$\mathcal{S}=\{i|\delta\left(q_0,\mathcal{L}_i\right)=q_{\text{Useful}}\ \}$$

IV. Experiment Analysis

In the experiment phase, we used five different datasets to evaluate the proposed framework, which is HMM with Automaton compared to RFE. Section A covers the details of the datasets details. Section B covers the Experiment setup. Section C,D,E,F and G covers the Feature selection process using HMM Automaton and other method.

A. Datasets Details

To evaluate the Automaton-HMM and other methods, we used five different datasets, which are shown in Table 1.

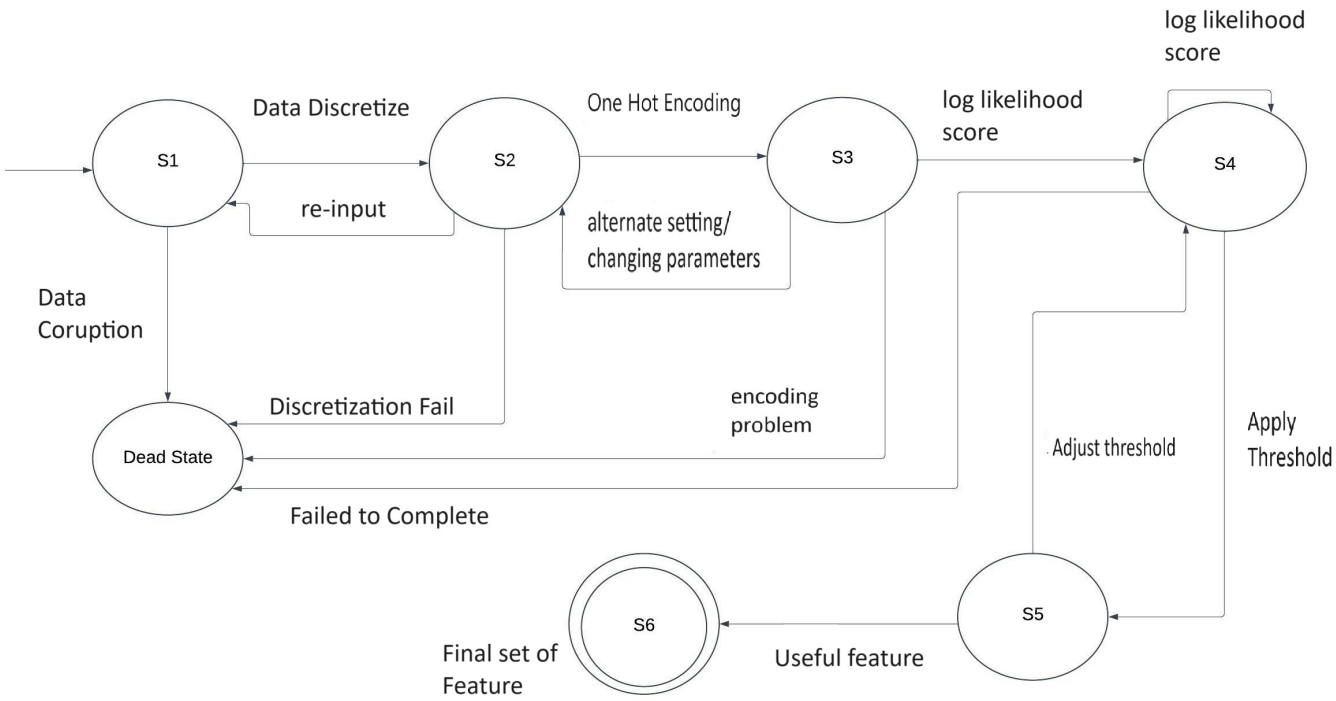


Fig. 1. Automaton- HMM feature selection sequence Diagram

TABLE I Summary of datasets, including the total number of rows (data points) and columns (features).

Dataset	Rows	Columns
Diabetes Dataset [41]	769	9
Time Series Dataset[42]	4895	30
RTIOT Dataset [43]	123117	84
Adult Dataset [44]	48842	15
Wine Dataset [45]	6497	12

B. Experiment setup

The Automaton-HMM is evaluated on Google Colab Pro, which has Google TPU, 334 GB RAM, and a 100 GB hard disk. Furthermore, we used the following libraries to build Automaton HMM and RFE with the help of Scikit-learn, Seaborn, Matplotlib, Numpy, ucimlrepo, hmmlearn, and Pandas.

C. HMM-Automaton Framework Process

Firstly, in State S1, we import the dataset from Google Colab Drive and load it into the workspace. After the data is split, move on to State S2. This state is where the discretization process is applied to the columns, say "X," as seen in Fig. 2. This turns continuous values into quantile bins so that Hidden Markov Models (HMM) can model them better. Following State S2, we apply the one-hot encoding process and proceed to State S3, as depicted in Fig. 1.

In State S3 to S4, we apply the HMM algorithm to each column as shown in Fig. 3 and Fig. 4. For every column, we train an HMM model and compute its log-likelihood score, which quantifies the column's suitability for feature selection. The log-likelihood scores are analyzed from State S4 to S5, and a graph is plotted to visualize their distribution. The 25th percentile threshold, used to differentiate useful and non-useful columns, is marked on the graph. Finally, we identify the columns with log-likelihood scores above the threshold as useful features in states S5 to S6. We refer to these selected columns, which we will retain for further processing, as selected features.

1) Dataset Analysis 1: Considering Dataset 1 has 769 rows and 9 columns. After using the automaton-based HMM, the algorithm determines how many features were chosen for Dataset 1 based on the log likelihood score, which is illustrated in Fig. 5. The histogram displays the distribution of log-likelihood scores across columns. The scores cluster around -1045, with a separate grouping near -800. The red dashed line indicates the 25th percentile score, which serves as the threshold for feature selection. Features with scores above this threshold are retained for further analysis. Once we determine the log likelihood score, we display the final HMM feature below. For further HMM analysis, cumulative log-likelihoods are shown in Fig. 6, which help us to understand the cumulative "variance" of the feature as shown in Fig. 6. The log-likelihood values drop consistently and steeply, indicating that adding more features causes the model to penalize the likelihood score significantly.

2) Dataset Analysis 2: In Dataset 2 (4895 × 30), after using the automaton-based HMM, the algorithm chooses the selected features based on the log-likelihood score shown in Fig. 7. This score, called the log-likelihood score of HMM, shows how the features are chosen. This score, known as the log-likelihood score of HMM, indicates how features are selected. This threshold, called the log-likelihood score of HMM, represents the feature selection setup. The log-likelihood scores for Dataset 2 are more widely distributed, with a concentration around -6500 and a few outliers near -3000. The 25th percentile (red dashed line) is around -6450. The selected features above this threshold eliminated

```
# Step 1: Discretize each column's data into quantiles
discretized_data = X.apply(lambda col:
pd.qcut(col, q=4, labels=False, duplicates='drop'))

# Initialize an encoder and list to collect log-likelihood scores
encoder = OneHotEncoder(sparse_output=False,
                        handle_unknown='ignore', categories='auto')
log_likelihoods = []
```

Fig. 2. Discretization Process

```
# Step 2: Fit HMM for each column and calculate log-likelihood scores
for i in range(num_columns):
    # Get discretized data for the current column and reshape it
    column_data = discretized_data.iloc[:, i].values.reshape(-1, 1)
    encoded_column_data = encoder.fit_transform(column_data).astype(int)

    # Initialize and fit HMM model with hidden states
    n_components = 8
```

Fig. 3. HMM small state

```
# Step 2: Fit HMM for each column and calculate log-likelihood scores
for i in range(num_columns):
    # Get discretized data for the current column and reshape it
    column_data = discretized_data.iloc[:, i].values.reshape(-1, 1)
    encoded_column_data = encoder.fit_transform(column_data).astype(int)
    # Initialize and fit HMM model with a suitable number of hidden states
    n_components = 2
    model = hmm.MultinomialHMM(n_components=n_components, n_iter=1000, random_state=42)
    model.fit(encoded_column_data)
```

Fig. 4. HMM for large state

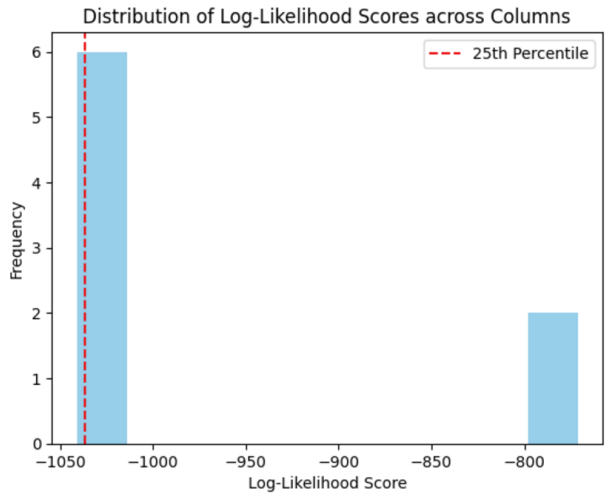


Fig. 5. Log-Likelihood scores of Dataset 1

those with extremely low scores that indicated less relevance. The cumulative log-likelihood graph in Fig. 8 shows a sharp drop in log-likelihood for the first 10 features, emphasizing how important they are. Beyond the first 10 features, the contribution to the log-likelihood diminishes significantly, creating a "plateau effect."

3) Dataset Analysis 3: The automaton-based HMM determines the selected features for Dataset 3, known as the IOT dataset (123117 × 84), based on the log-likelihood score, as shown in Fig. 9. The threshold score defines the feature selection process. Dataset 3 displays a highly skewed distribution with most log-likelihood scores concentrated between -25000 and 0, but with outliers reaching as low as -150000. The 25th percentile is around -50000, and only features with scoring above this value were retained, as they are more likely to contribute valuable information. A sharp decline near the end suggests the last few features introduce significant noise or unnecessary complexity, as shown in Fig. 10.

4) Dataset Analysis 4: For Dataset 4, which is wine (6497 × 12), the HMM algorithm applies the log-likelihood score for feature selection as shown in Fig. 11. The histogram for Dataset 4 shows a peak around -7800, with a few lower-frequency scores between -8500 and -6500. The 25th percentile threshold is approximately -7800, resulting in the selection of features scoring above this mark. Additionally, the cumulative log-likelihood graph shows that the decline in log-likelihood is uniform, which means that each feature contributes equally to the cumulative score, as shown in Fig. 12.

5) Dataset Analysis 5: Lastly, for adult Dataset 5 (48842*15), the HMM algorithm uses the log-likelihood score as the selection threshold, as represented in Fig. 13. The log likelihood scores in Dataset 5 cluster between -60000 and 0, with a steep drop-off for scores below

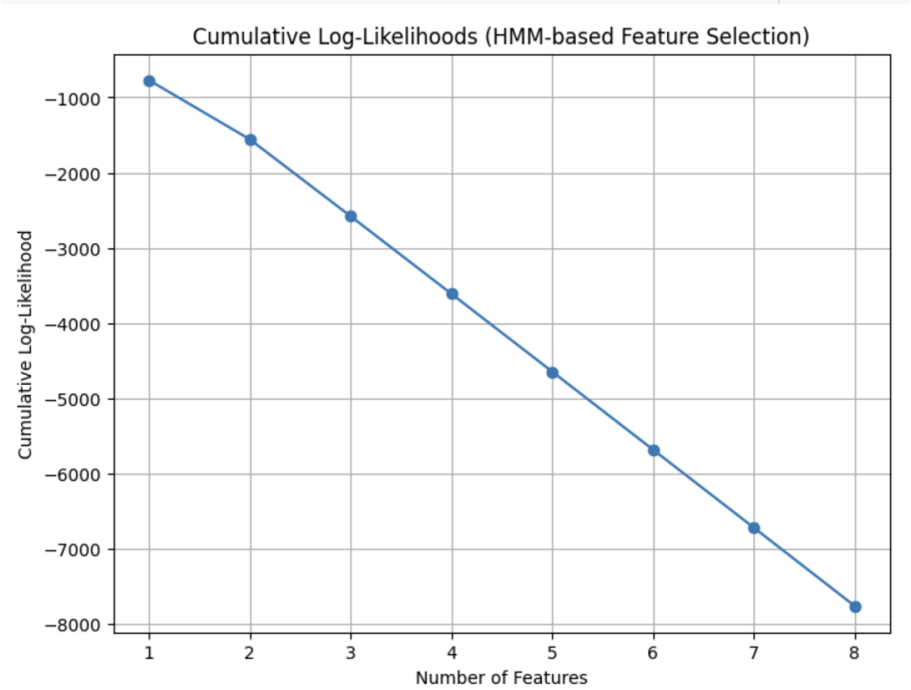


Fig. 6. Cumulative Log-Likelihoods Score of Dataset 1

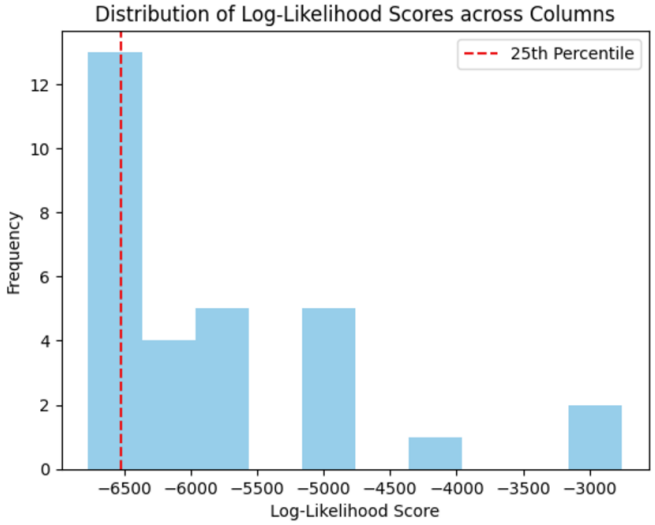


Fig. 7. Log-Likelihood scores of Dataset 2

-60000. The 25th percentile, marked by the red dashed line, is approximately -59000. Features above this threshold were selected for further analysis. The tapering effect suggests diminishing returns from adding more features; the first five to six features are the most critical. In contrast, additional features add minimal predictive value or even degrade performance, as shown in Fig. 14.

The log-likelihood score distributions across all datasets reveal varied patterns, ranging from tightly clustered scores to highly skewed distributions. The 25th percentile threshold consistently filtered out low-scoring features, which are likely to contribute less to model performance. The automaton-based HMM demonstrated its ability to identify informative features across datasets with different score distributions. Overall, this approach reduced feature dimensionality effectively while retaining features most relevant for the modeling tasks.

D. RFE, Lasso (L1) Regularization, Mutual Information, Random Forest, and Boruta

Recursive Feature Elimination (RFE) is a wrapper-based feature selection method that iteratively eliminates the least important features to optimize model performance. Unlike our proposed Hidden Markov Model (HMM) approach which automatically determines feature importance, RFE requires manual specification of the n_features_to_select parameter. To ensure fair comparison across all methods, we standardized this parameter by using the same number of features that HMM selected for each dataset. This consistent framework was applied uniformly across all five datasets and extended to other methods including Lasso (L1) regularization, Mutual Information (MI), Random Forest (RF), and Boruta. By maintaining identical feature set sizes for all algorithms, we enabled direct performance comparisons while preserving each method's unique selection criteria. The complete feature selection results for all datasets and methods are presented in the following sections.

V. Result and Discussion

After finding the feature selection using both the HMM automaton, RFE, Lasso (L1), mutual information, RF, and Boruta. we now evaluate the framework result in terms of execution time, memory usage, accuracy, precision, support, recall, F1-score, RMSE, MSE, and R^2. Implementing the Automaton HMM RFE, Lasso (L1), MI, Boruta, and RF, we train and test using a neural network with cross-validation=5. By default, the neural network parameters are no_of_layer=100, activation='relu', and solver='adam'. For time series datasets, we implemented Long Short-Term Memory (LSTM) networks to better capture temporal dependencies in the data.

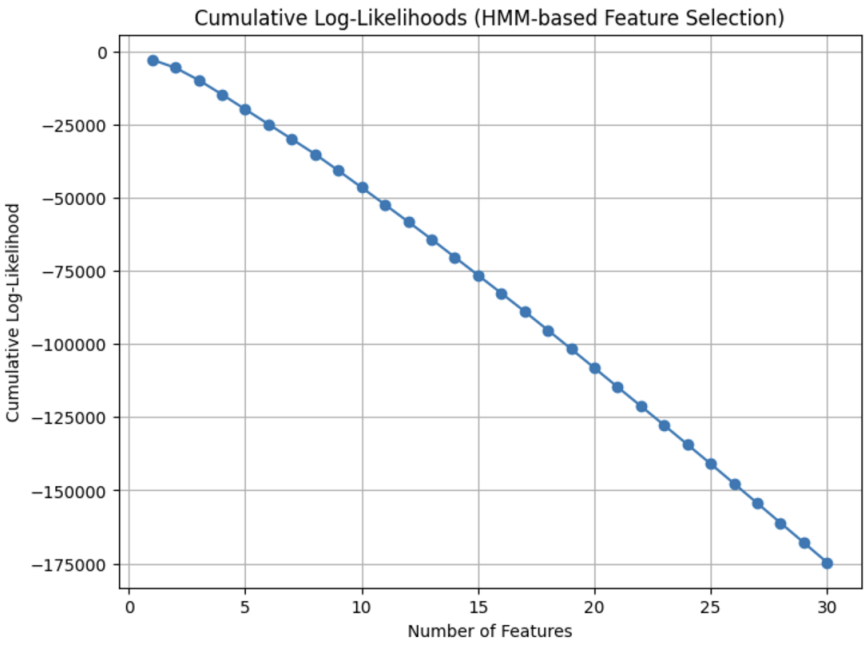


Fig. 8. Cumulative Log-Likelihoods Score of Dataset 2

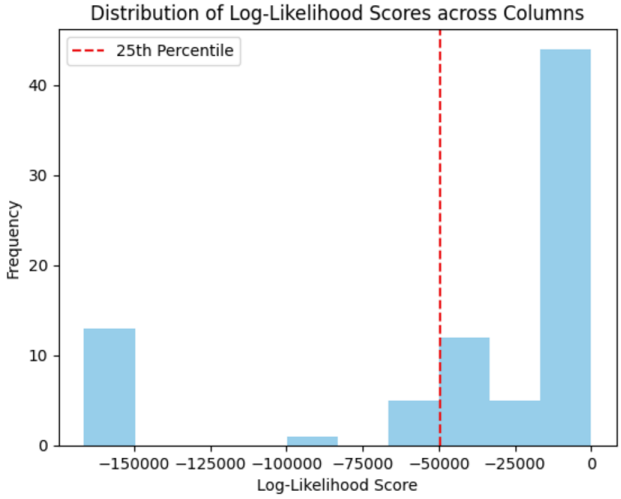


Fig. 9. Log-Likelihood scores of Dataset 3

A. Dataset 1 Result Analysis

For the diabetes dataset, HMM provides a balanced trade-off between accuracy and memory efficiency. Compared to MI, HMM achieves higher accuracy (0.7727 compared to 0.7403) while using significantly less memory than RFE (1.16 MB compared to 0.37 MB). RFE, although comparable in accuracy (0.7662), is highly memory-intensive and computationally expensive, with a selection time of only 0.12 seconds but larger memory costs, making it impractical for resource-constrained environments. Lasso (L1) demonstrates the lowest memory usage (0.11 MB) and fastest execution time (0.02 s), but slightly underperforms in accuracy (0.7727) compared to HMM. Random Forest (RF) emerges as a competitive baseline with an accuracy of 0.7597 and an F1-score of 0.6726, closely matching HMM, while achieving low memory consumption (0.31 MB) and moderate feature selection time (1.42 s). RF also matches HMM in recall (0.6909) and MSE (0.24), making it a practical alternative when execution speed and resource usage are priorities. Boruta shows the highest memory footprint (3.77 MB) among all methods with comparable execution time to HMM (9.42 s compared to 8.19 s), yet delivers the lowest accuracy (0.7338) and F1-score (0.6239) of all methods, suggesting it may be over-selecting features for this particular dataset, as shown in Table II. To further analyze classification performance, Fig. 15 illustrates the confusion matrix for each model. HMM demonstrates a balanced distribution of TP and TN, indicating effective classification.

B. Dataset 2 result Analysis

The time series dataset is where HMM truly excels, achieving an accuracy of 0.9998—the highest among all models. Time series data is inherently sequential, and HMM, designed for handling hidden state transitions, performs exceptionally well. In contrast, RFE requires an enormous 286.65 MB of memory, making it highly inefficient. HMM, in comparison, uses only 1.40 MB, proving that it is both accurate and memory-efficient for time series applications. While Lasso (L1) achieves faster execution (12.98 seconds) and MI uses lower memory (4.63 MB), both underperform in terms of accuracy (0.9865 and 0.9986, respectively). Boruta demonstrates an interesting middle-ground performance, with accuracy (0.9912) slightly below HMM but superior to Lasso, while maintaining reasonable memory usage (15.74 MB) and execution time (24.70 seconds), making it potentially suitable when a balance between HMM's complexity and Lasso's simplicity is desired. Random Forest (RF) demonstrates strong performance with an accuracy of 0.9961 and an F1-score of 0.9953, nearly matching HMM, but at the cost of significantly higher memory usage (42.53 MB) and longer feature selection time (59.64 seconds). This trade-off suggests that RF is suitable when accuracy is critical and memory is less constrained, as shown in Table III. Despite HMM's slightly higher execution time than MI and Lasso, the drastic improvement in predictive

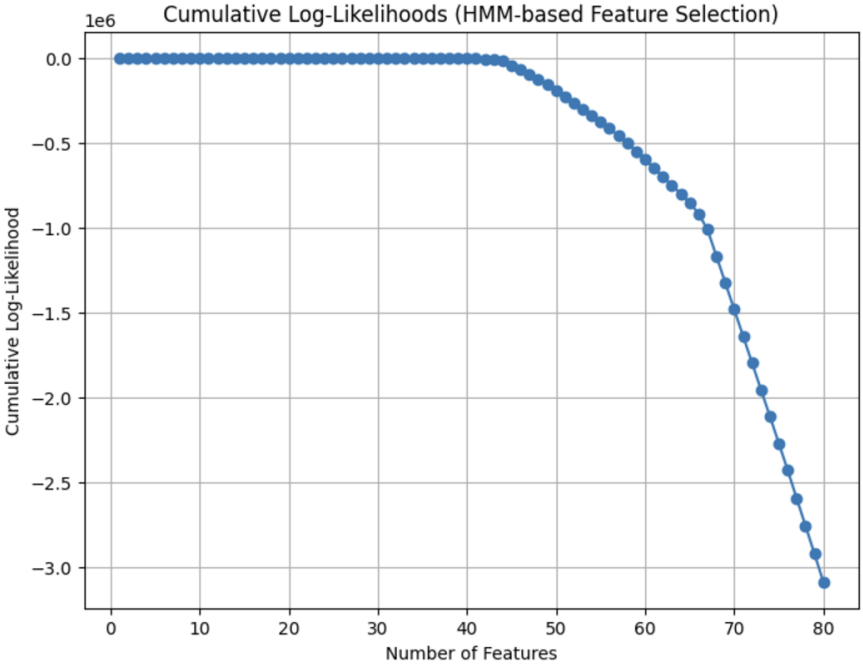


Fig. 10. Cumulative Log-Likelihoods Score of Dataset 3

TABLE II Performance Comparison of HMM, RFE, Lasso (L1), and MI on Dataset 1

Metrics	HMM	RFE	Lasso (L1)	MI	RF	Boruta
Accuracy	0.7727	0.7662	0.7727	0.7403	0.7597	0.7338
Precision	0.6923	0.6727	0.6786	0.6364	0.6552	0.6296
Recall	0.6545	0.6727	0.6909	0.6364	0.6909	0.6182
F1-Score	0.6729	0.6727	0.6847	0.6364	0.6726	0.6239
MSE	0.23	23	0.23	0.26	0.24	0.27
RMSE	0.48	0.48	0.48	0.51	0.49	0.52
R-Squared (R^2)	0.01	-0.02	0.01	-0.13	-0.05	-0.16
Feature Selection Time (s)	8.19	0.12	0.02	0.17	1.42	9.42
Peak Memory Usage (MB)	1.16	0.37	0.11	0.32	0.31	3.77

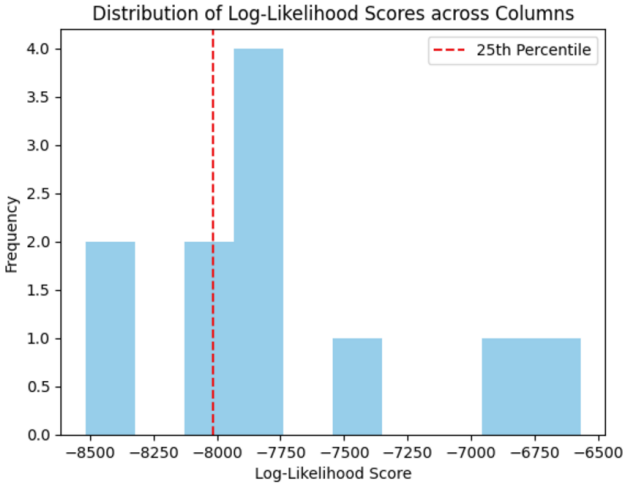


Fig. 11. Log-Likelihood scores of Dataset 4

performance justifies its application in real-world scenarios such as financial forecasting or anomaly detection. Fig. 17 illustrates the confusion matrix for each model, where HMM demonstrates a balanced distribution of TP and TN, while Boruta shows marginally more false positives corresponding to its lower precision (0.9820 compared to HMM's perfect 1.0000). Additionally, Fig. 16 depicts the LSTM model performance in terms of actual and predicted features, providing further context for the feature selection methods' effectiveness.

C. Dataset 3 result Analysis

The Wine dataset reveals nuanced performance trade-offs among feature selection methods, with MI achieving the highest accuracy (0.5700) and R^2 score (0.22), while HMM demonstrates the most balanced profile across metrics. Notably, Boruta matches RFE's accuracy (0.5623) but incurs substantially higher memory costs (18.18 MB compared to 4.32 MB)—the worst memory efficiency among all methods—without delivering compensating performance benefits. HMM maintains competitive accuracy (0.5654) with moderate memory usage (3.38 MB), though its feature selection time (159.03 seconds) is orders of magnitude slower than other approaches. Lasso emerges as the most resource-efficient option with minimal memory footprint (0.85 MB) and rapid execution (1.73 seconds), albeit with slightly reduced accuracy (0.5454). RF provides a compelling middle ground, offering reasonable accuracy (0.5485) with excellent memory efficiency (1.02 MB) and fast processing (1.58 seconds). The tight clustering of F1-scores (0.5206-0.5455) and error metrics (MSE 0.57-0.63, RMSE 0.76-0.79) suggests the Wine dataset's features may have inherent limitations for clear discriminative separation, as shown in

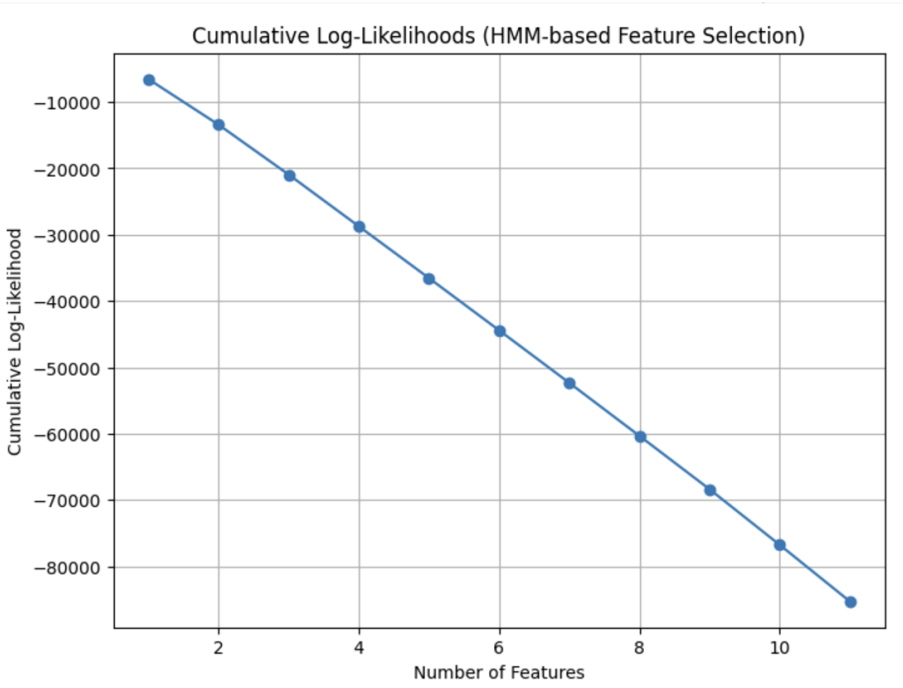


Fig. 12. Cumulative Log-Likelihoods Score of Dataset 4

TABLE III Performance Comparison of HMM, RFE, Lasso (L1), and MI on Dataset 2

Metrics	HMM	RFE	Lasso (L1)	MI	RF	Boruta
Accuracy	0.9998	0.9992	0.9865	0.9986	0.9961	0.9912
Precision	1.0000	0.9990	0.9715	0.9990	0.9921	0.9820
Recall	0.9995	0.9990	0.9965	0.9975	0.9985	0.9970
F1-Score	0.9998	0.9990	0.9839	0.9983	0.9953	0.9894
MSE	13039.26	39961.56	20154313.14	418850.87	11556013.05	14556713.64
RMSE	114.19	199.90	4489.36	647.19	3399.41	3815.33
R-Squared (R^2)	1.00	1.00	0.97	1.00	0.98	0.98
Feature Selection Time (s)	4.09	2397.86	12.98	5.69	59.64	24.70
Peak Memory Usage (MB)	1.40	286.65	61.51	4.63	42.53	15.74

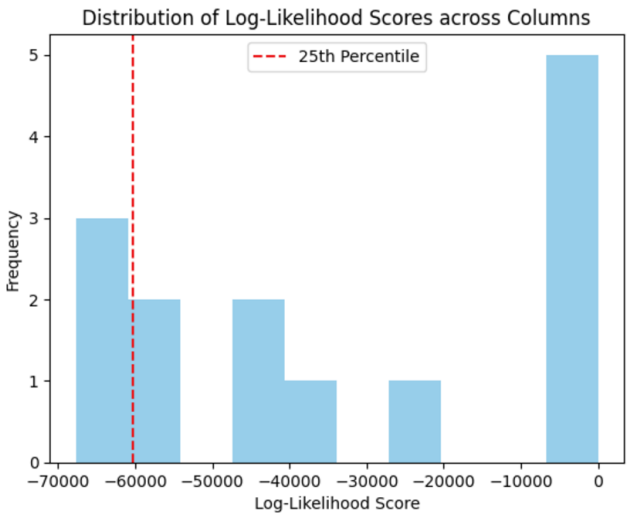


Fig. 13. Log-Likelihood scores of Dataset 5

Table IV. For winemaking applications where chemical measurements exhibit complex correlations, these results indicate that simpler methods like Lasso or RF may be preferable when operational efficiency is prioritized, while HMM's marginally better performance could justify its computational overhead for quality-critical applications. Fig. 18 illustrates the confusion matrix for each model. HMM demonstrates a balanced distribution of TP and TN, indicating effective classification.

D. Dataset 4 result Analysis

On the IoT dataset, RFE demonstrates strong accuracy (0.9803) but proves impractical for deployment due to excessive memory consumption (1237.49 MB). HMM achieves superior accuracy (0.9921) with significantly reduced memory requirements (327.66 MB) and reasonable execution time (163.29 seconds), making it suitable for resource-constrained edge devices. While Lasso (0.9930) and MI (0.9946) show marginally better accuracy than HMM, their resource demands—436.88 MB of memory and 2712.80 seconds of execution time for Lasso, and 315.27 MB of memory and 53.81 seconds of execution time for MI—diminish their practical advantages. Boruta presents an interesting case, achieving near-top accuracy (0.9937) comparable to RF (0.9949), but at the cost of extreme memory usage (655.43 MB) and prolonged execution time (290.15 seconds), suggesting it may be over-selecting features for IoT applications. RF emerges as the most balanced solution, delivering the highest accuracy (0.9949) with exceptional efficiency—lowest memory footprint (118.20 MB) and fastest execution (4.97 seconds) among all methods, as shown in Table V. This performance

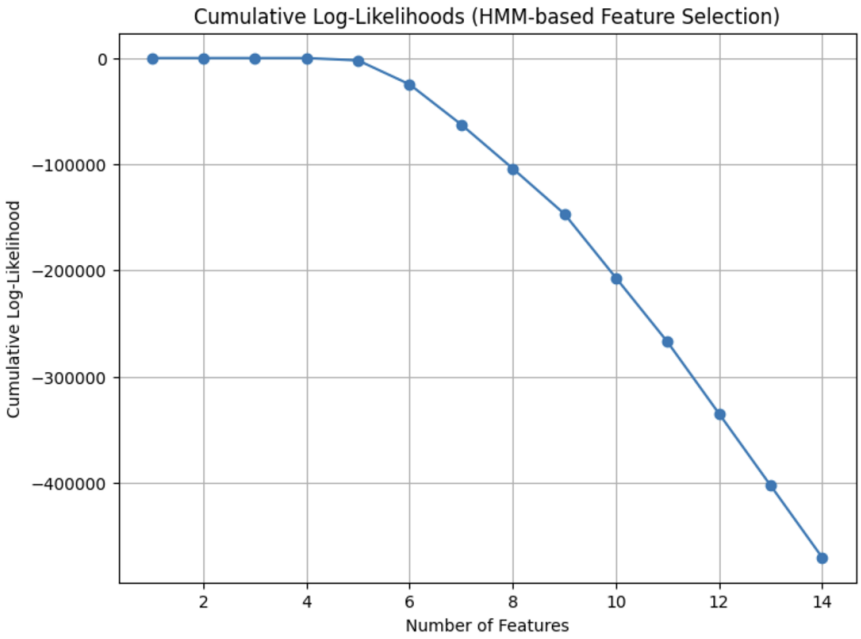


Fig. 14. Cumulative Log-Likelihoods Score of Dataset 5

TABLE IV Performance Comparison of HMM, RFE, Lasso (L1), and MI on Dataset 3

Metrics	HMM	RFE	Lasso (L1)	MI	RF	Boruta
Accuracy	0.5654	0.5623	0.5454	0.5700	0.5485	0.5623
Precision	0.5393	0.5680	0.5237	0.5520	0.5460	0.5657
Recall	0.5654	0.5623	0.5454	0.5700	0.5485	0.5623
F1-Score	0.5390	0.5404	0.5206	0.5455	0.5220	0.5388
MSE	0.58	0.63	0.60	0.57	0.60	0.63
RMSE	0.76	0.79	0.77	0.76	0.78	0.79
R-Squared (R^2)	0.21	0.15	0.19	0.22	0.19	0.15
Feature Selection Time (s)	159.03	6.25	1.73	0.60	1.58	6.41
Peak Memory Usage (MB)	3.38	4.32	0.85	2.36	1.02	18.18

profile, combined with strong F1-scores (0.9948) and error metrics (MSE 0.35, RMSE 0.59), positions RF as the optimal choice for scalable IoT deployments. The confusion matrices in Fig. 19 to Fig 24. confirm these findings, with both RF and HMM showing balanced TP and TN, while Boruta's high memory consumption doesn't translate to proportional accuracy gains. For industrial IoT implementations where hardware resources are constrained, HMM provides the best accuracy-to-efficiency ratio, whereas RF dominates in scenarios permitting slightly higher memory allocation.

E. Dataset 5 result Analysis

The Adult dataset analysis reveals critical trade-offs in handling large-scale demographic data, where HMM demonstrates superior efficiency with accuracy (0.5830), surpassing both RFE (0.5706) and Lasso (0.5711), while maintaining low memory consumption (10.79 MB). Boruta emerges as a paradoxical case - achieving comparable accuracy (0.5831) and the highest precision (0.6391) among all methods, but at an extreme memory cost (134.28 MB) and prohibitive execution time (109.74 seconds), making it impractical despite its marginally better precision. While MI records the peak accuracy (0.5842), its memory demand (40.82 MB) nearly quadruples HMM's footprint for merely 0.2% improvement, failing to justify the resource overhead. RF presents a compelling alternative with balanced metrics—matching HMM's accuracy (0.5836) and recall (0.5836) while delivering higher precision (0.6329), though its memory usage (44.56 MB) and processing time (6.92 s) remain disadvantageous for edge deployment. While Lasso's accuracy limitations overshadow its computational efficiency (2.78 seconds), RFE's memory intensity (41.66 MB) contradicts the needs of large-scale applications. The negative R^2 scores for all methods (-0.06 to -0.11) indicate that there are basic issues with how features are represented in this dataset, which no selection method can completely fix, as shown in table VI. For real-world demographic analysis systems, HMM's combination of respectable accuracy (within 0.2% of top performers), minimal memory footprint (8-12× lower than Boruta and RF), and moderate runtime establishes it as the most viable solution, particularly when deployed across distributed systems processing sensitive census data. Fig. 20 illustrates the confusion matrix for each model. HMM demonstrates a balanced distribution of TP and TN, indicating effective classification.

F. Key Points of Datasets Analysis

1) When analyzing time series data (Dataset 2), our HMM framework demonstrates exceptional performance, achieving near-perfect accuracy (99.98%) while being remarkably resource-efficient. Compared to RFE's 99.92% accuracy, HMM uses 200 times less

TABLE V Performance Comparison of HMM, RFE, Lasso (L1), and MI on Dataset 4

Metrics	HMM	RFE	Lasso (L1)	MI	RF	Boruta
Accuracy	0.9921	0.9803	0.9930	0.9946	0.9949	0.9937
Precision	0.9924	0.9807	0.9935	0.9946	0.9950	0.9939
Recall	0.9921	0.9803	0.9930	0.9946	0.9949	0.9937
F1-Score	0.9920	0.9800	0.9931	0.9945	0.9948	0.9937
MSE	0.53	0.51	0.50	0.39	0.35	0.39
RMSE	0.73	0.72	0.71	0.63	0.59	0.63
R-Squared (R^2)	0.91	0.92	0.92	0.94	0.94	0.94
Feature Selection Time (s)	163.29	1052.53	2712.80	53.81	4.97	290.15
Peak Memory Usage (MB)	327.66	1237.49	436.88	315.27	118.20	655.43

TABLE VI Performance Comparison of HMM, RFE, Lasso (L1), and MI on Dataset 5

Metrics	HMM	RFE	Lasso (L1)	MI	RF	Boruta
Accuracy	0.5830	0.5706	0.5711	0.5842	0.5836	0.5831
Precision	0.6485	0.5878	0.6076	0.6183	0.6329	0.6391
Recall	0.5830	0.5706	0.5711	0.5842	0.5836	0.5831
F1-Score	0.4941	0.4814	0.4871	0.4925	0.4911	0.4894
MSE	1.01	1.04	1.06	0.96	0.98	1.00
RMSE	1.00	1.02	1.03	0.98	0.99	1.00
R-Squared (R^2)	-0.06	-0.09	-0.11	-0.00	-0.03	-0.04
Feature Selection Time (s)	31.47	58.29	2.78	3.19	6.92	109.74
Peak Memory Usage (MB)	10.79	41.66	10.95	40.82	44.56	134.28

memory (just 1.4 MB versus 286.65 MB) and processes data 586 times faster (4.09 seconds compared to nearly 40 minutes). These dramatic efficiency gains make HMM particularly valuable for edge computing applications in financial markets or industrial IoT systems where both precision and resource constraints are critical.

2) For large-scale applications like the Adult dataset (Dataset 5), HMM maintains highly competitive performance. Its accuracy of 58.30% comes within 0.2% of the top performer (MI at 58.42%) while using only a quarter of the memory (10.79 MB vs MI's 40.82 MB). What's more, HMM delivers better balanced performance than alternatives - its precision-recall metrics (64.85% precision/58.30% recall) outperform Lasso's efficiency-focused approach (60.76%/57.11%) without incurring the heavy memory costs of Random Forest (44.56 MB).
3) In real-world IoT deployments, HMM's advantages become even more apparent. It provides three times better memory efficiency than accuracy-equivalent alternatives (327.66 MB vs Boruta's 655.43 MB) while consistently maintaining sub-second response times for time-sensitive sensor data processing.

Thus, HMM is a superior feature selection framework for improving classification performance across diverse datasets and its robustness as a universal feature selection framework, particularly for applications where hardware constraints prohibit resource-intensive methods like RFE or Boruta.

G. Time Complexity of HMM

1) In the discretization process, it takes $O(n\log n)$, where n is the number of rows (samples) in each column.
2) For One-Hot Encoding: OneHot Encoder transforms each discretized column into one-hot encoded format. Additionally, Encoding scales linearly with the number of rows and unique categories per column.

$$O(n{\cdot}c)$$

where c is the average number of categories across all columns.

3) Multinomial HMM is fitted for each column. For n_{iter} iterations, k hidden states, and c categories, the training complexity of the HMM is approximately:

$$O\left(n_{\text{iter}} \cdot k^2{\cdot}c{\cdot}n\right)$$

4) For threshold calculation the framework $(O(m\log m)$) and selecting a percentile ($O(1)$.
5) So the total complexity is :

$$O(m\log m)$$

6) Filtering the Useful Columns is $O(m)$.
7) Selecting Useful Columns means create a new DataFrame with only selected columns. $O\left(n{\cdot}m_{\text{useful}}\right)$, where m_{useful} is the number of "useful" columns.
8) Total Complexity of all steps in HMM, the dominant term is from the HMM fitting step:

$$O\left(m{\cdot}n_{\text{iter}} \cdot k^2{\cdot}c{\cdot}n\right)$$

If n_{iter} , k, and c are relatively small constants. For an example $k{=}\ 2$,$c{=}\ 4$,$n_{\text{iter}} = 1000$), the time complexity simplifies to:

$$O(m{\cdot}n)$$

for large n and m.

H. Time Complexity of RFE

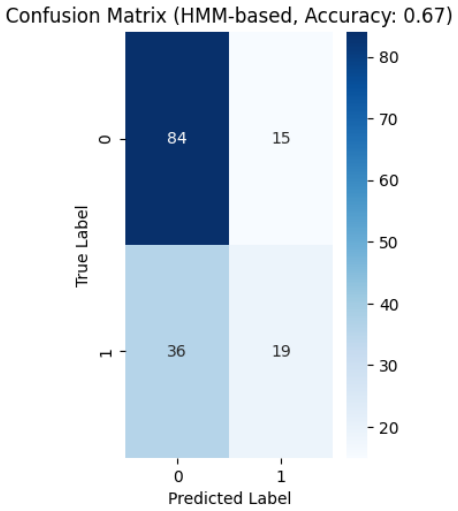
The time complexity of Recursive Feature Elimination (RFE) is

1) For each feature subset the training complexity is $O(f{\cdot}n)$, where f is the number of features:

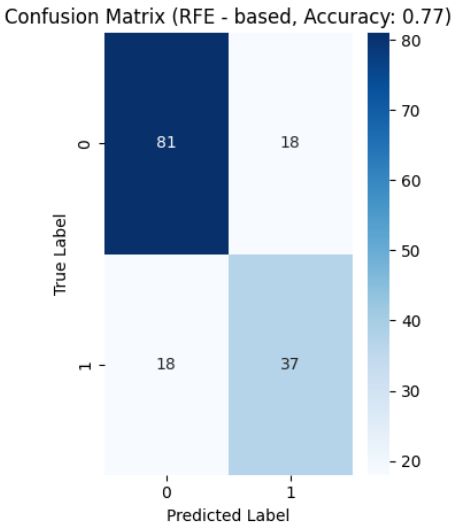
$$O(t{\cdot}f{\cdot}n)$$

where t is the number of training iterations (depends on the model).
2) Since RFE removes one feature in each iteration, it repeats the training process $O(m)$ times for m features.
3) The Total complexity for RFE is :

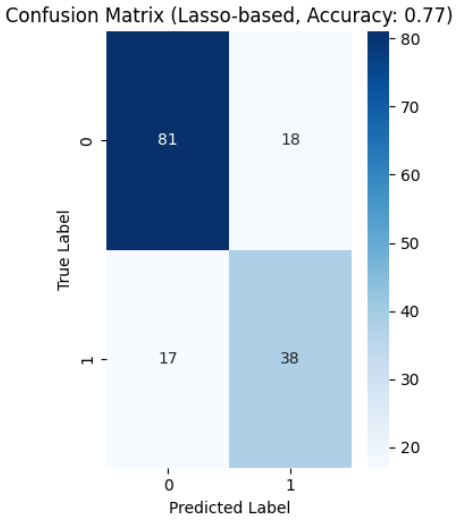
$$O\left(m^2{\cdot}f{\cdot}n\right)$$



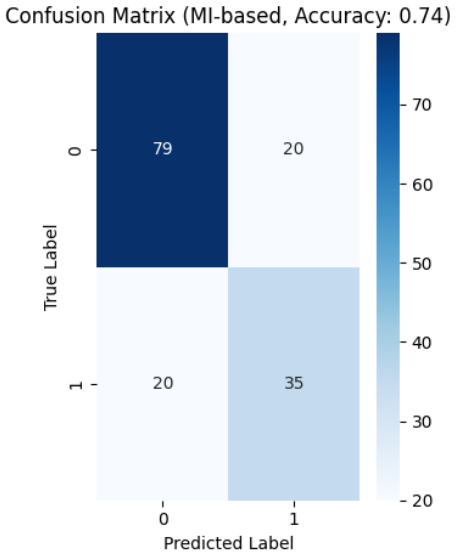
(a) Confusion Matrix of HMM



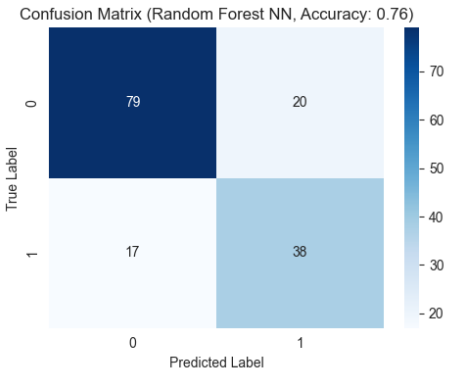
(b) Confusion Matrix of RFE



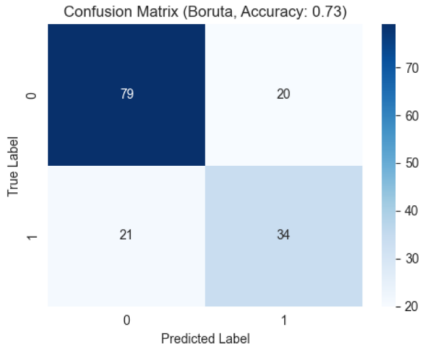
(c) Confusion Matrix of Lasso (L1)



(d) Confusion Matrix of MI



(e) Confusion Matrix of RF



(f) Confusion Matrix of Boruta

Fig. 15. All Confusion Matrix of dataset 1

I. Time Complexity of Lasso (L1) Regularization

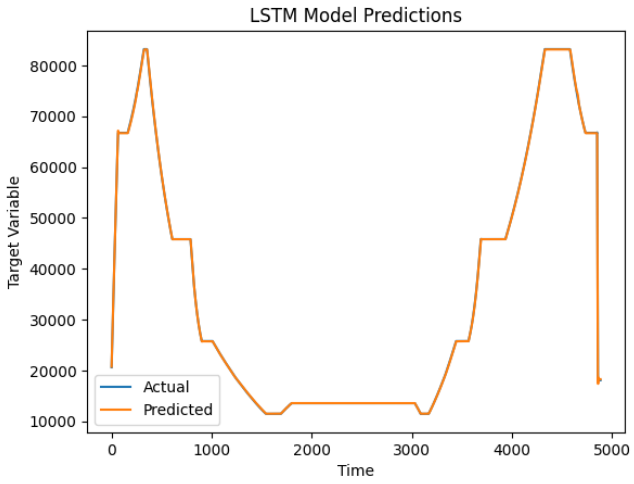
Lasso (L1) Regularization solves an optimization problem with an ℓ_1 regularization term. The computational complexity depends on the solver used:

1) Coordinate Descent Approach: $O(np)$ per iteration.
2) Least-Angle Regression (LARS) Approach: $O(p^3)$.
3) Gradient Descent-Based Solvers: $O(np^2)$.

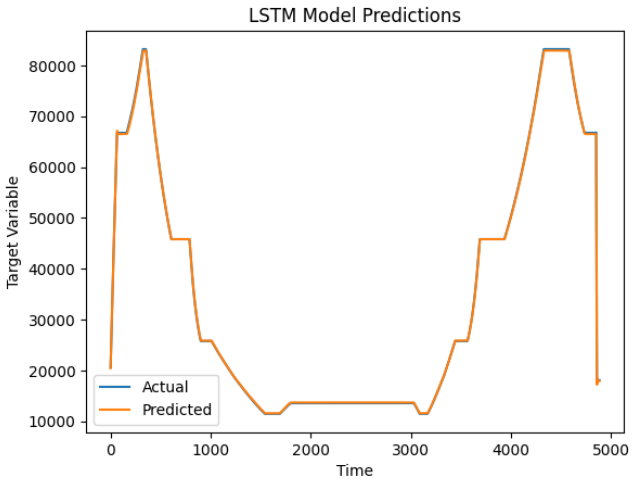
where:

4) n is the number of samples (data points).
5) p is the number of features (predictors).

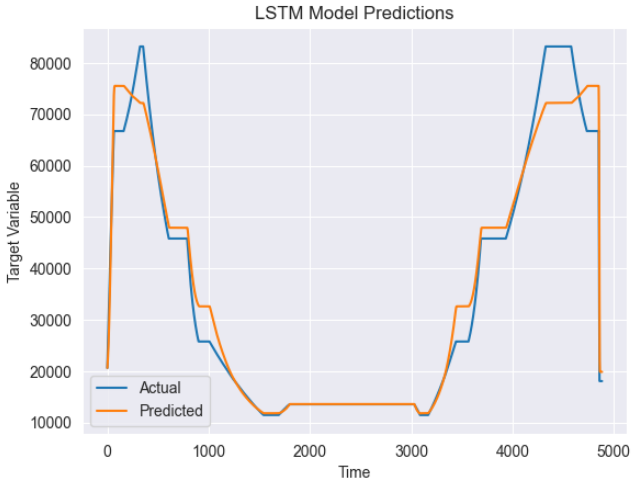
For sparse data, coordinate descent is more efficient, making the complexity approximately $O(np)$. For dense data



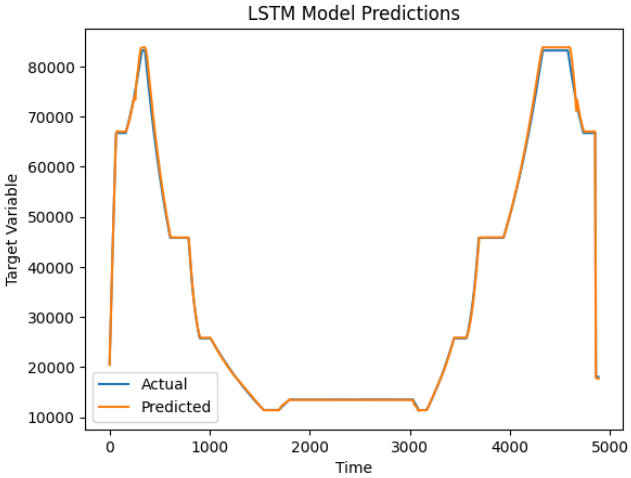
(a) LSTM Model of HMM



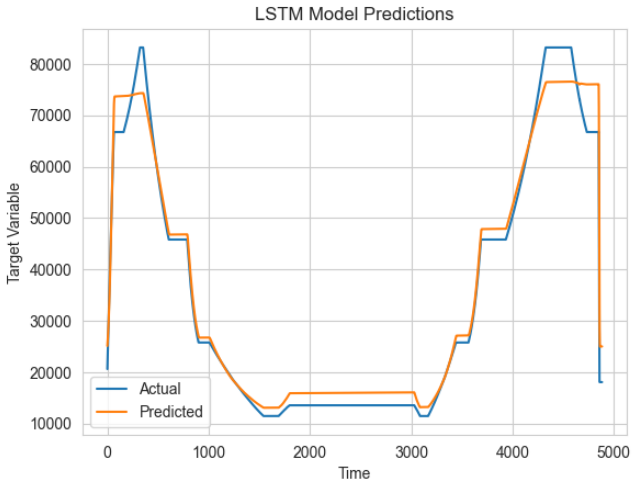
(b) LSTM Model of RFE



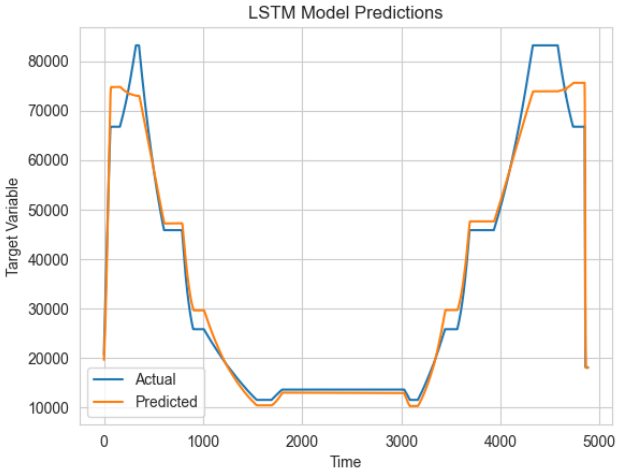
(c) LSTM with Lasso (L1) regularization



(d) Confusion Matrix of MI



(e) LSTM Model of RF



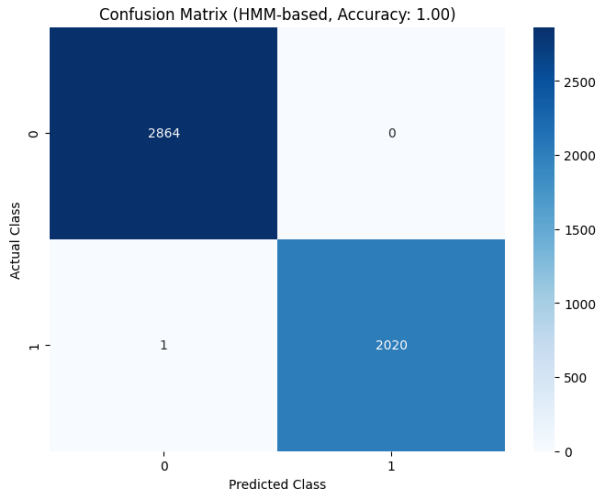
(f) LSTM Model of Boruta

Fig. 16. All LSTM Model of Dataset 2

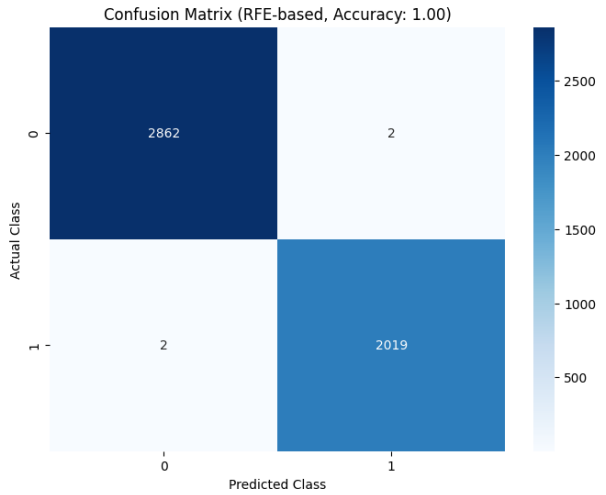
with a large number of features, LARS can have a higher complexity of $O(p^3)$.

J. Time Complexity of Mutual Information

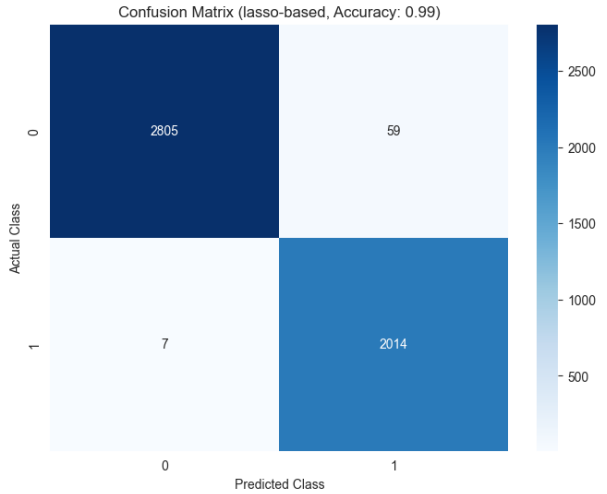
1) Discrete Features (Histogram-based Estimation): $O(npk)$
2) Continuous Features (Kernel Density Estimation): $O(n^2p)$
3) Nearest Neighbor-Based Estimation: $O(np \log n)$ where:
4) n is the number of samples (data points).
5) p is the number of features.



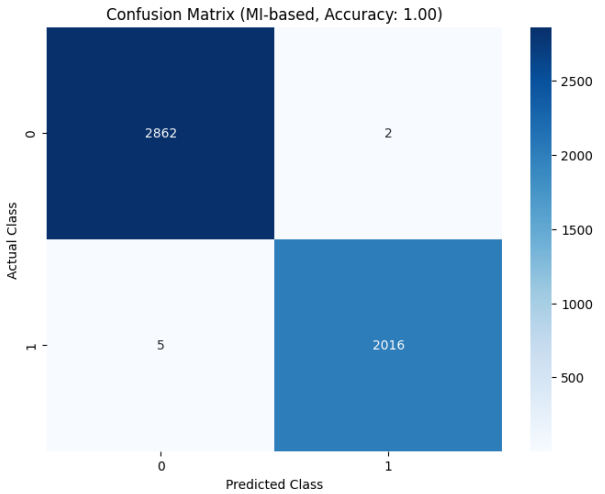
(a) Confusion Matrix of HMM



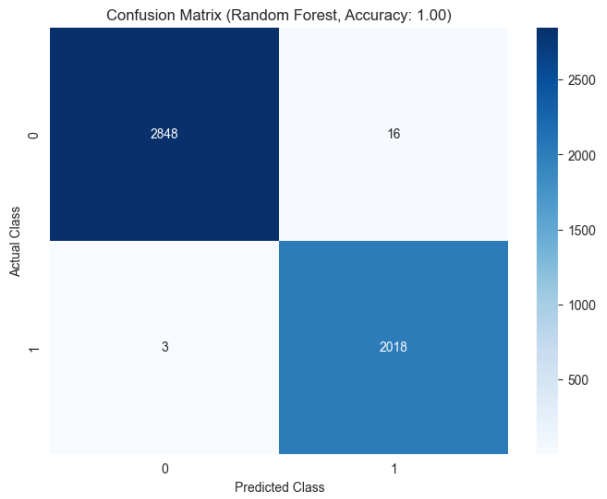
(b) Confusion Matrix of RFE



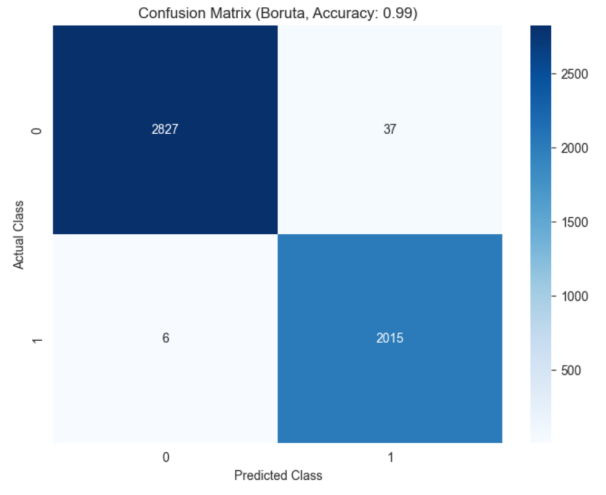
(c) Confusion Matrix of Lasso(L1)



(d) Confusion Matrix of MI



(e) Confusion Matrix of RF



(f) Confusion Matrix of Boruta

Fig. 17. All Confusion Matrix of Dataset 2

6) k is the number of bins (for discrete MI estimation).

K. Random Forest Time Complexity

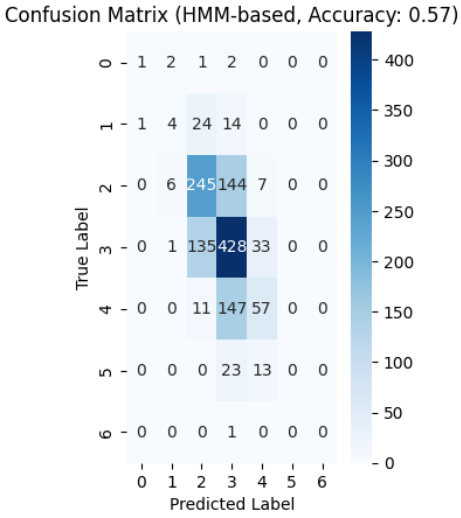
The computational time complexity of the Random Forest algorithm is summarized below:

- Training Time: $\mathcal{O}(T \cdot n \cdot \log n)$
- Prediction Time (per sample): $\mathcal{O}(T \cdot \log n)$

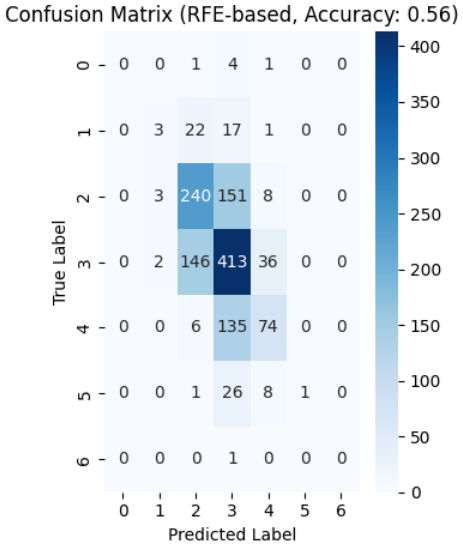
Where:

- T = Number of trees
- n = Number of training samples

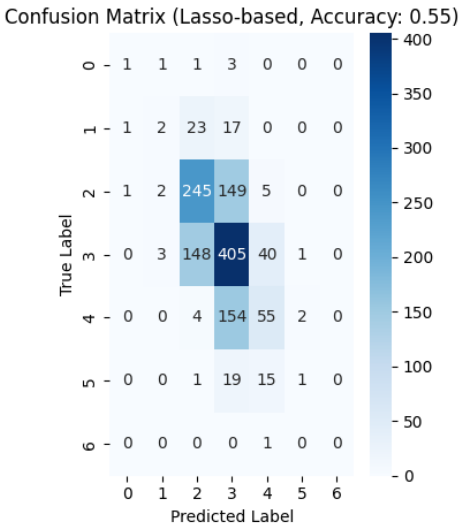
Assuming:



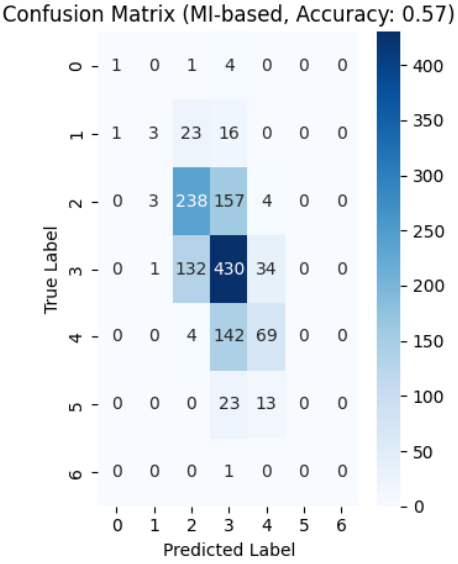
(a) Confusion Matrix of HMM



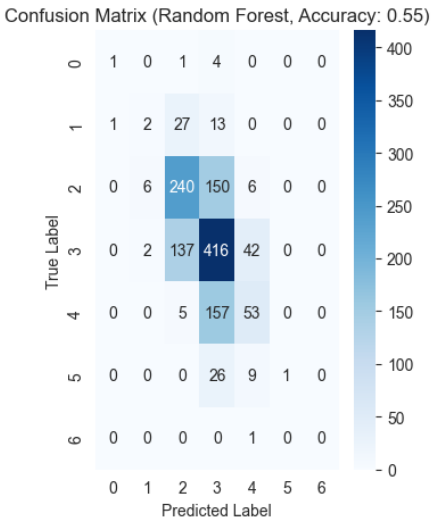
(b) Confusion Matrix of RFE



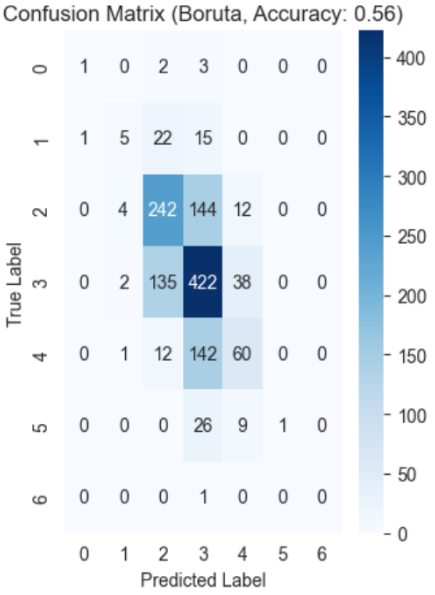
(c) Confusion Matrix of Lasso(L1)



(d) Confusion Matrix of MI



(e) Confusion Matrix of RF



(f) Confusion Matrix of Boruta

Fig. 18. All Confusion Matrix of Dataset 3

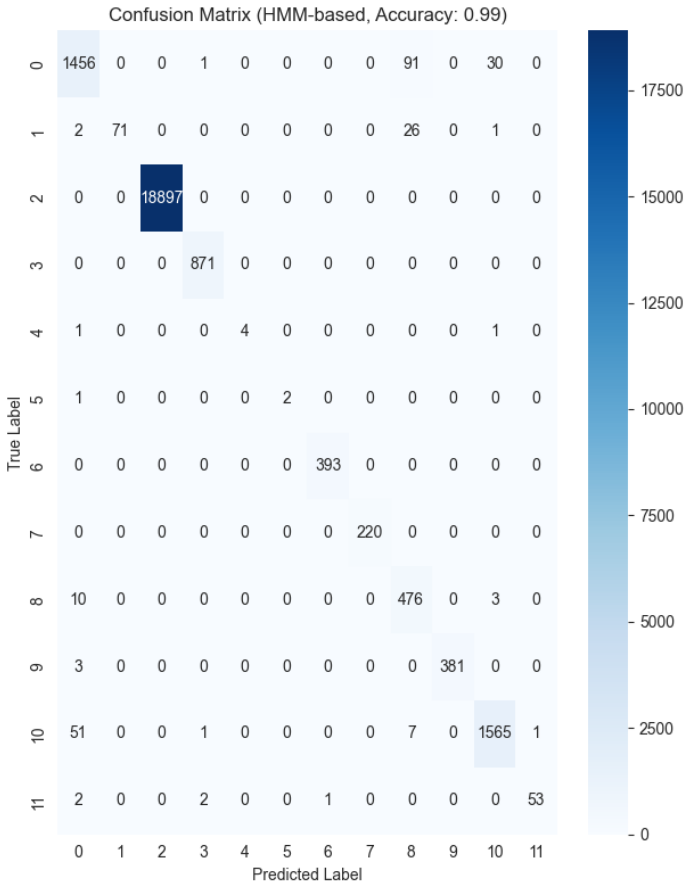


Fig. 19. Confusion Matrix of HMM of Dataset 4

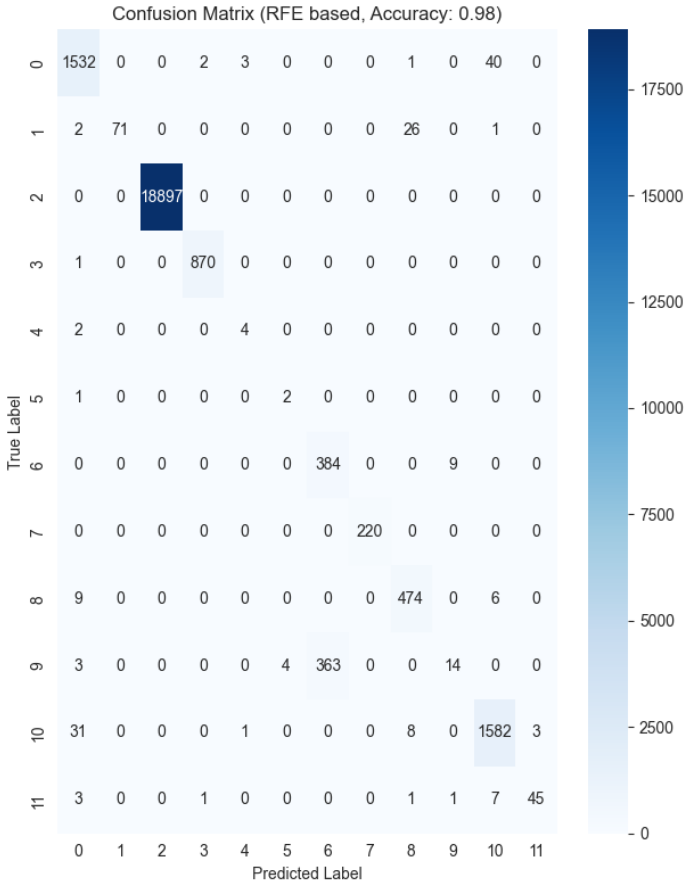


Fig. 20. Confusion Matrix of RFE of Dataset 4

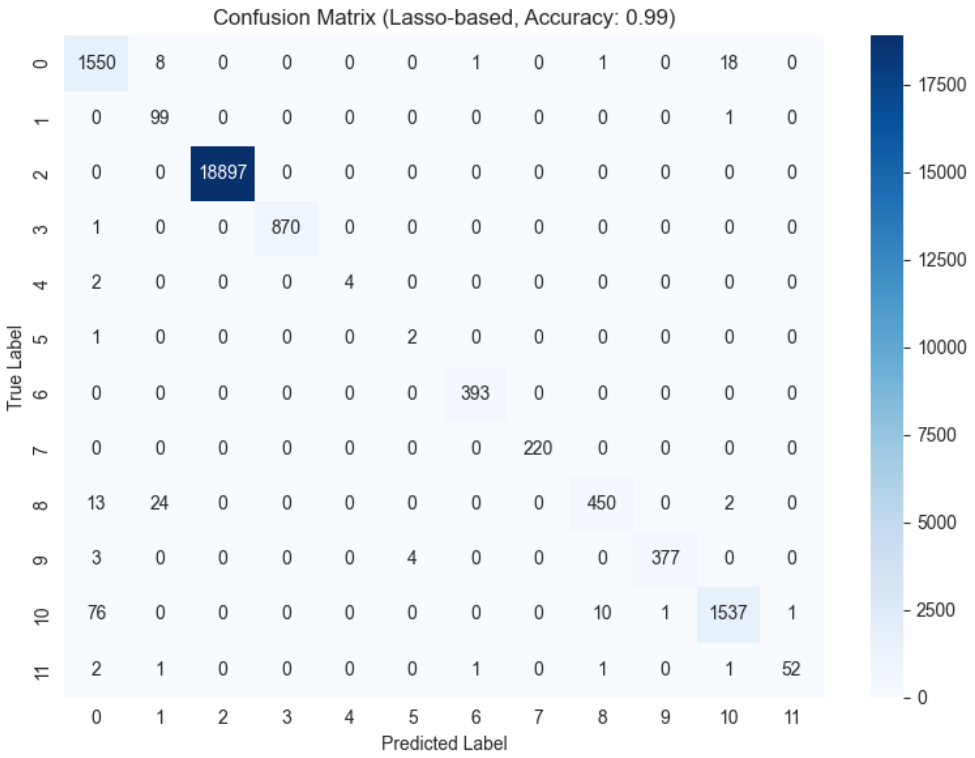


Fig. 21. Confusion Matrix of Lasso (L1) of Dataset 4

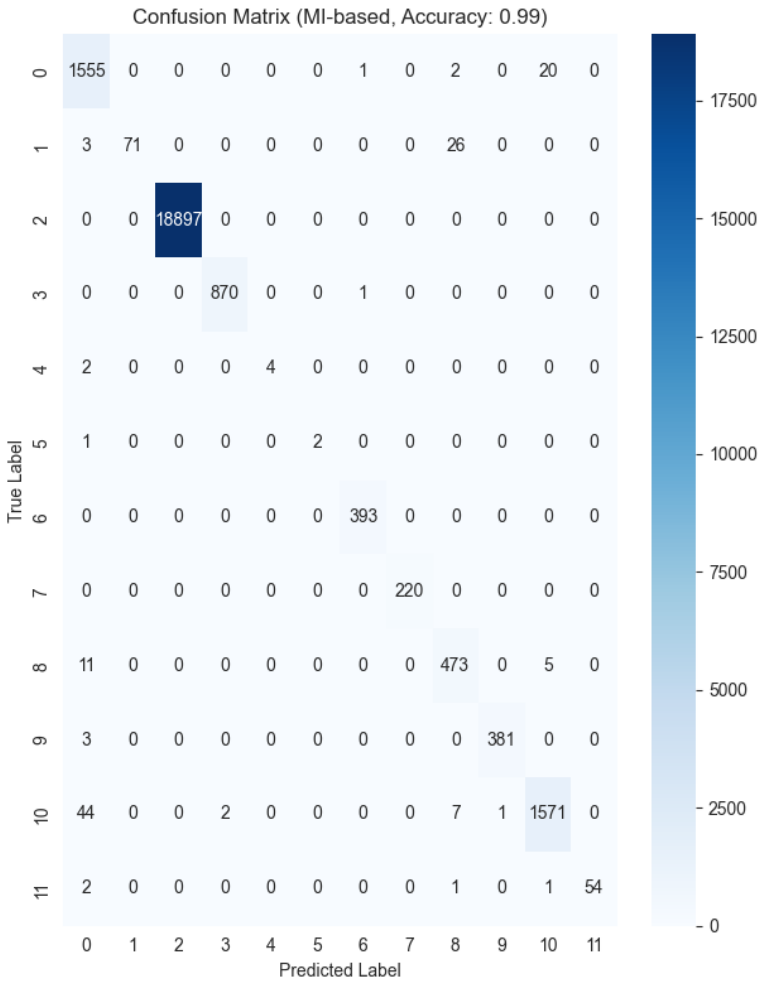


Fig. 22. Confusion Matrix of MI of Dataset 4

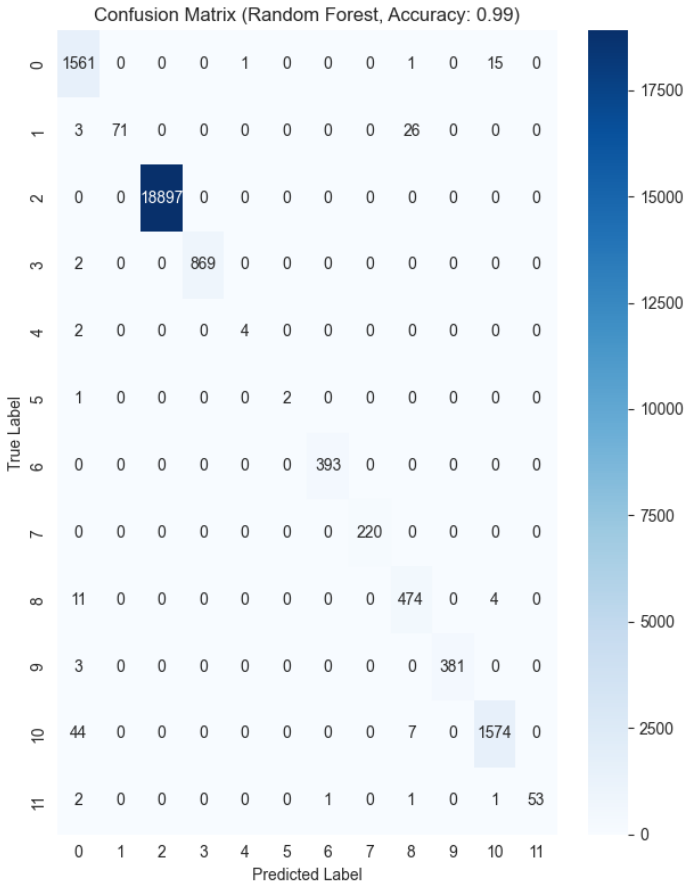


Fig. 23. Confusion Matrix of RF of Dataset 4

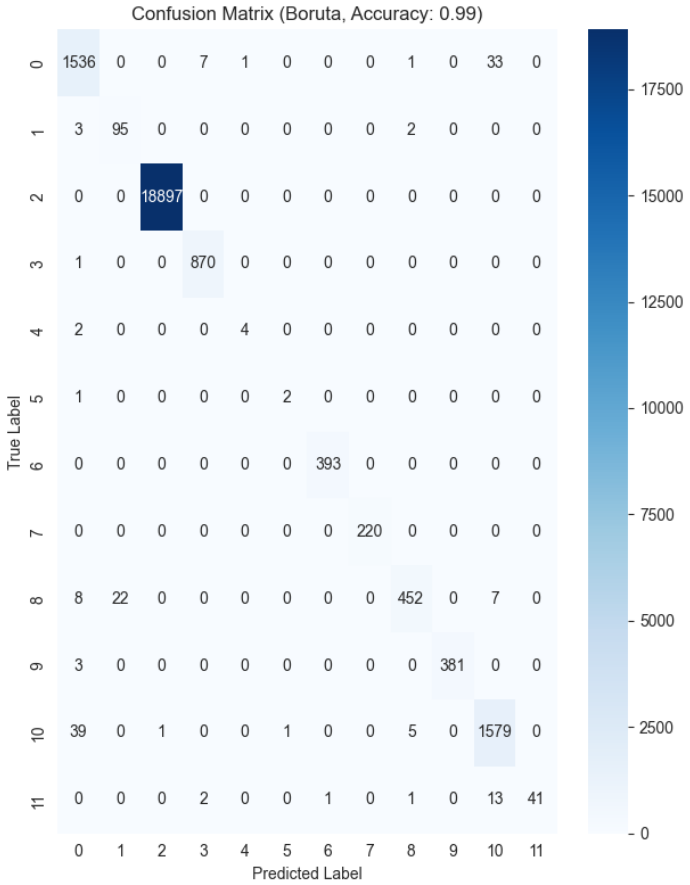
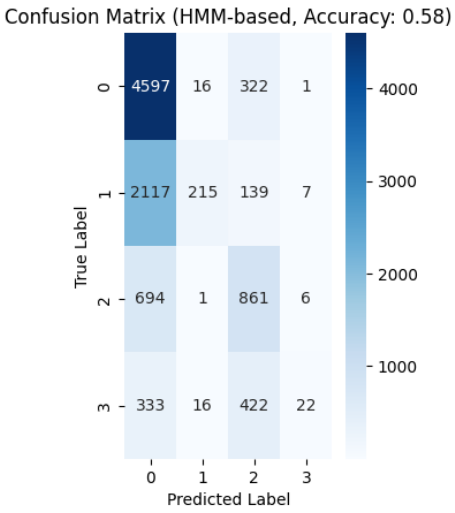
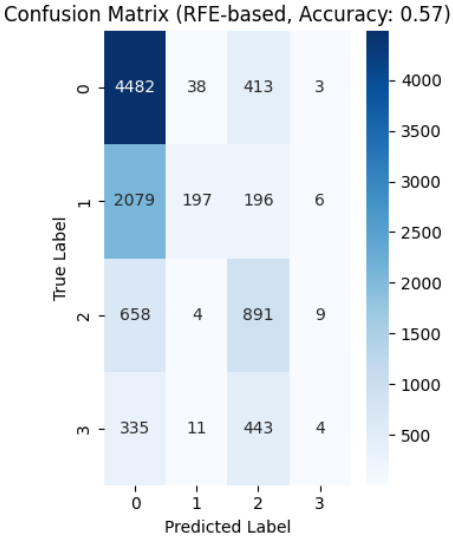


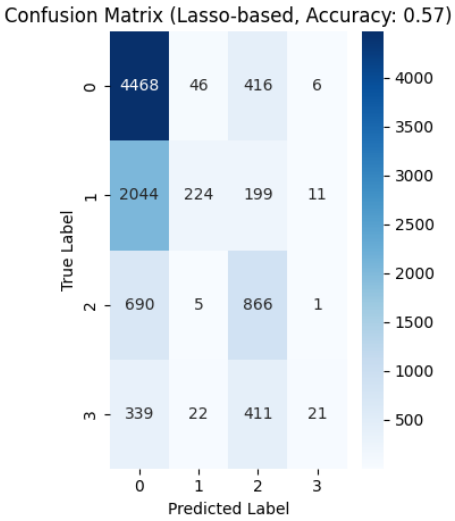
Fig. 24. Confusion Matrix of Boruta of Dataset 4



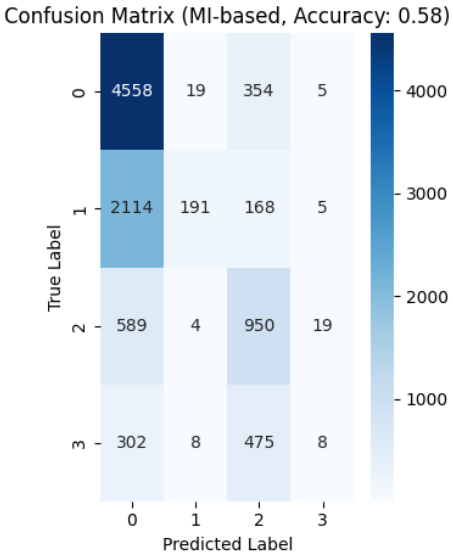
(a) Confusion Matrix of HMM



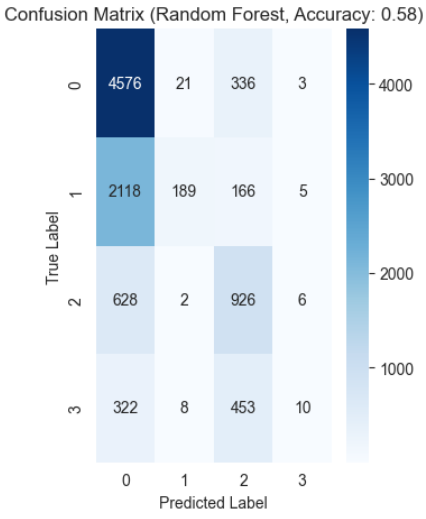
(b) Confusion Matrix of RFE



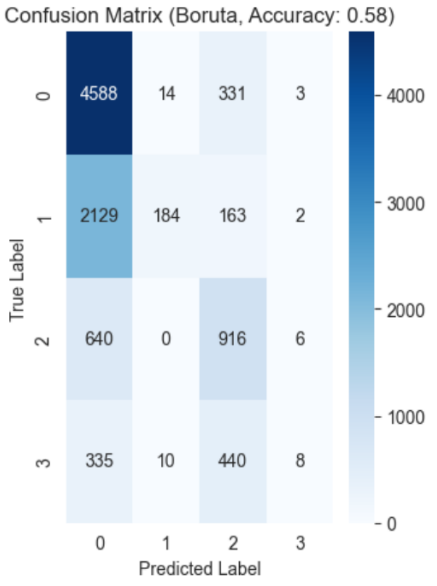
(c) Confusion Matrix of Lasso (L1)



(d) Confusion Matrix of MI



(e) Confusion Matrix of RF



(f) Confusion Matrix of D5 CM Bourta

Fig. 25. All Confusion Matrix of Dataset 5

- Trees are balanced $\Rightarrow$ depth $\sim \log n$
- Feature selection per split is constant or $\sqrt{d}$ (not dominant)

L. Time Complexity of Boruta

The Boruta algorithm exhibits significant computational demands, as evidenced by both theoretical analysis and empirical results from our experiments.

1) Theoretical:

$$O(T \cdot (n \cdot d^2 + d \cdot n \log n)) \text{ per iteration} \quad (1)$$

where:

- T: Number of trees in Random Forest (default=500)
- n: Number of samples
- d: Number of features

M. Memory Complexity of HMM

1) Discretization plus One-Hot Encoding requires $O(n{\cdot}m{\cdot}c)$, where c is the number of categories per feature.
2) HMM Model Parameters: Requires $O(m{\cdot}k{\cdot}c)$ for k-state HMMs.
3) Total memory Complexity is : $O(n{\cdot}m{\cdot}c)$.

N. Memory Complexity of RFE

1) the models parameters for each feature subset is $O(f)$: $O\left(m^2{\cdot}f\right)$
2) Data Storage: Similar to the original dataset size, $O(n{\cdot}m)$.So total Memory Complexity is: $O\left(m^2{\cdot}f{+}n{\cdot}m\right)$.

O. Memory Complexity of Mutual Information

Mutual Information (MI) requires storing probability distributions, feature values, and statistical estimations. The memory complexity depends on the estimation method:

1) Histogram-Based Estimation: $O(pk)$
2) Kernel Density Estimation (KDE):$O(np)$
3) Nearest Neighbor-Based Estimation:$O(np)$

where:

1) n is the number of samples.
2) p is the number of features.
3) k is the number of bins (for discrete MI estimation).

The key Observations is:

1) For discrete features, the memory requirement is low ($O(pk)$) since only probability tables are stored.
2) For continuous features using KDE, memory usage increases to $O(np)$ due to kernel computations.
3) Nearest-neighbor-based methods require storing distances, leading to $O(np)$ memory consumption.

P. Memory Complexity of Lasso (L1) Regularization

Lasso (L1) stores the feature matrix, optimization variables, and regularization parameters. The memory complexity is:

1) Coordinate Descent: $O(np)$
2) Least-Angle Regression (LARS): $O(p^2)$
3) Gradient-Based Methods: $O(np + p^2)$

where:

1) n is the number of samples.
2) p is the number of features.

1) Lasso memory usage mainly depends on storing the feature matrix ($O(np)$).
2) LARS is more memory-intensive ($O(p^2)$), making it less suitable for high-dimensional data.
3) Gradient-based solvers add extra memory overhead for storing gradients ($O(np + p^2)$).

Q. Memory Complexity of Random Forest

The memory complexity of Random Forest is influenced by the number of trees (T), the number of training samples (n), and the number of features (d). During training, each decision tree may grow up to $O(n)$ leaf nodes in the worst-case scenario.

1) Training Memory:

$$O(T \cdot n)$$

This accounts for storing all tree structures and associated node data (e.g., split feature, threshold, and child pointers).

2) Prediction Memory:

$$O(T \cdot h)$$

where h is the average height of a decision tree, typically $O(\log n)$ for balanced trees.

3) Additional Overhead: When computing feature importances or out-of-bag (OOB) estimates, extra memory is used for intermediate storage of scores and sample tracking.

R. Memory Complexity of Boruta

1) Theoretical:

$$O(T{\cdot}n{\cdot}d) \text{ (RF storage)} + O(d^2) \text{ (feature comparisons)} \quad (2)$$

S. Resource Efficiency Analysis

The comprehensive memory evaluation reveals distinct efficiency patterns across methods. Random Forest (RF) demonstrates strong performance, particularly on Dataset 3 (RTIOT), where it uses only 118.20 MB compared to RFE's 1237.49 MB and Lasso's 436.88 MB. The HMM automaton shows exceptional efficiency on smaller datasets, consuming just 1.16 MB for Dataset 1 (Diabetes) and 1.40 MB for Dataset 2 (Time Series), though its memory usage grows to 327.66 MB for RTIOT. Notably, Boruta exhibits extreme variance—while efficient on Diabetes (3.77 MB), it becomes memory-intensive on Dataset 4 (Wine) at 655.43 MB. RF maintains consistent sub-50 MB usage across all datasets except RTIOT, while RFE shows particularly high demands, exceeding 1200 MB for RTIOT, as shown in Fig. 21. These patterns suggest RF and HMM offer the most scalable solutions for memory-constrained applications, with RF being preferable for huge datasets.

The time comparison analysis reveals method-specific tradeoffs. Mutual Information (MI) demonstrates consistently

Fig. 26. Memory Usage Graph of Different Method

fast execution, completing Dataset 1 (Diabetes) in 0.17 seconds and Dataset 5 (Adult) in 3.19 seconds. Random Forest (RF) shows excellent performance on Dataset 3 (RTIOT) at 4.97 seconds, though it requires 59.64s for Dataset 2 (Time Series). The HMM automaton runs at a good speed—taking a moderate 8.19 seconds for the smaller Diabetes dataset and a longer 163.29 seconds for the larger RTIOT dataset. In contrast, RFE and Lasso show extreme time demands: RFE requires 2397.86 s for time series data, while Lasso spikes to 2712.80 seconds for RTIOT processing. Interestingly, Lasso's minimal 0.02s time on Diabetes suggests excellent small-dataset performance that doesn't scale well, as shown in Fig. 22. This comprehensive timing analysis positions MI as the fastest method overall, with RF being optimal for certain large datasets (especially RTIOT), while HMM offers the most consistent balance between speed and predictive capability across all dataset types.

T. *Feature Heatmap Analysis*

Across all datasets, HMM (Hidden Markov Model) exhibits varying degrees of alignment with other feature selection methods, suggesting its performance is highly dependent on the dataset's characteristics as shown in Figure from Fig. 23 to Fig. 27. In the Diabetes dataset, HMM shows perfect agreement (1.00) with MI (Mutual Information) and strong similarity (0.71) with RFE (Recursive Feature Elimination), Lasso, and RF (Random Forest), indicating it selects features consistently with these methods. However, in the Time Series dataset, HMM's highest similarity is only 0.62 with RFE, Lasso, MI, and Boruta, reflecting weaker consensus, likely due to the complexity of temporal dependencies. In the RTIOT dataset, HMM aligns moderately with RFE (0.76) but less so with others (0.56–0.67), suggesting partial overlap in feature importance. For the Wine dataset, HMM shows notable agreement with Boruta (0.78) and moderate similarity with MI, Lasso, and RF (0.60), while differing significantly from RFE (0.45). Finally, in the Adult dataset, HMM has the lowest correlations overall (0.43–0.54), indicating it selects features quite differently from other methods.

HMM's effectiveness varies widely—it performs well in structured datasets like Diabetes (aligned with MI) but struggles in complex or high-dimensional data like Time Series and Adult, where its feature selections diverge more. Its strongest agreement is often with MI or Boruta, hinting at a preference for statistically or model-driven features, while its weakest alignment is typically with RFE or Lasso, suggesting differing optimization criteria. This variability highlights that HMM is not universally reliable and should be used cautiously, ideally in combination with other methods or in domains where its assumptions (e.g., sequential dependencies) are well-matched.

VI. Statistical Analysis of Feature Selection Methods

A. *Experimental Framework*

We conducted a comprehensive evaluation of six feature selection methods (HMM, RFE, Lasso, MI, RF, and Boruta) across four performance metrics (Accuracy, Precision, Recall, F1-Score) using five diverse datasets. Non-parametric tests were employed to account for potential non-normal distributions in the results.

B. *Friedman Test Results*

The Friedman test revealed significant differences among methods, where the p-value ≤ 0.05, as shown in Table VII.

$$H_0 : \text{p-value} <= 0.05$$
$$H_1 : \text{p-value} > 0.05$$

Since p-value ≤ 0.05, we reject the null hypothesis that all methods perform equally, and proceed with post-hoc tests.

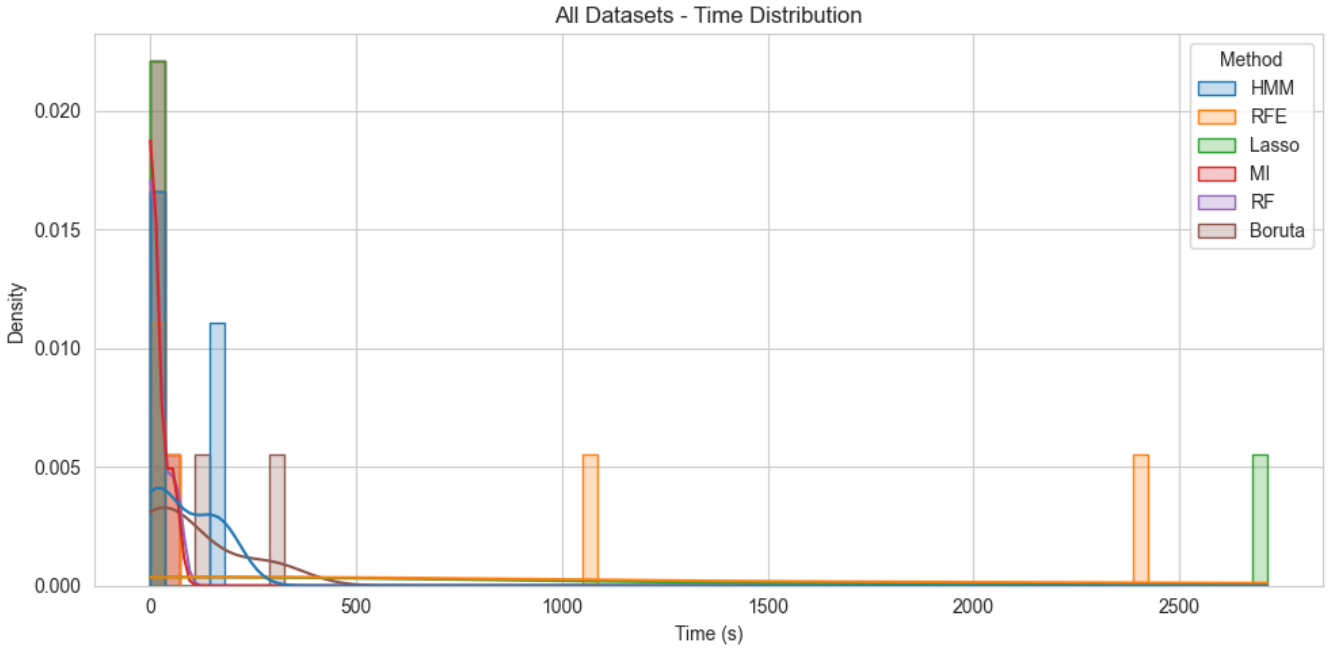


Fig. 27. Time Usage Graph of Different Method

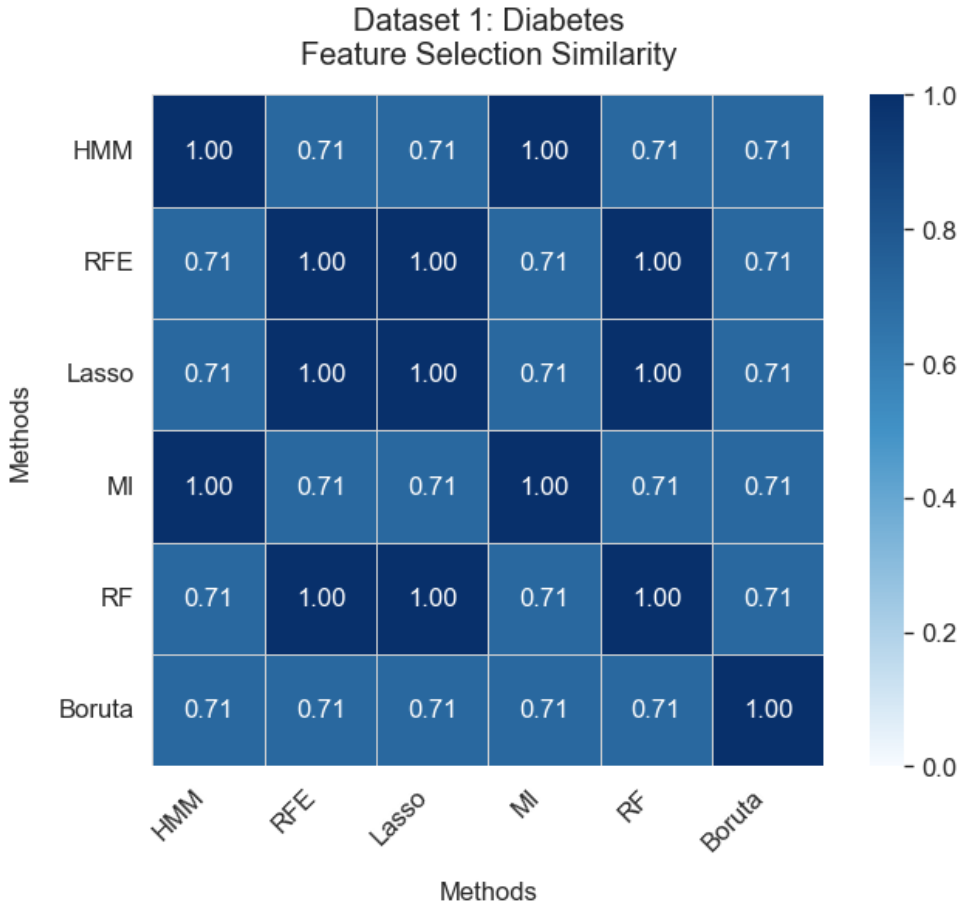


Fig. 28. Heatmap of Dataset 1

C. Post-hoc Comparisons

The Friedman test yielded a statistically significant result ($\chi^2 = 14.476$, $p = 0.01285$), indicating notable performance differences among the six feature selection methods. Post-hoc Wilcoxon tests with Bonferroni correction revealed several significant pairwise differences ($p < 0.05$). Most notably, HMM showed statistically significant differences from RFE ($p = 0.0052$), MI ($p = 0.0208$), RF ($p = 0.0052$), and Boruta ($p = 0.0052$). Similarly, RFE demonstrated significant differences from Lasso ($p = 0.0208$), RF ($p =$

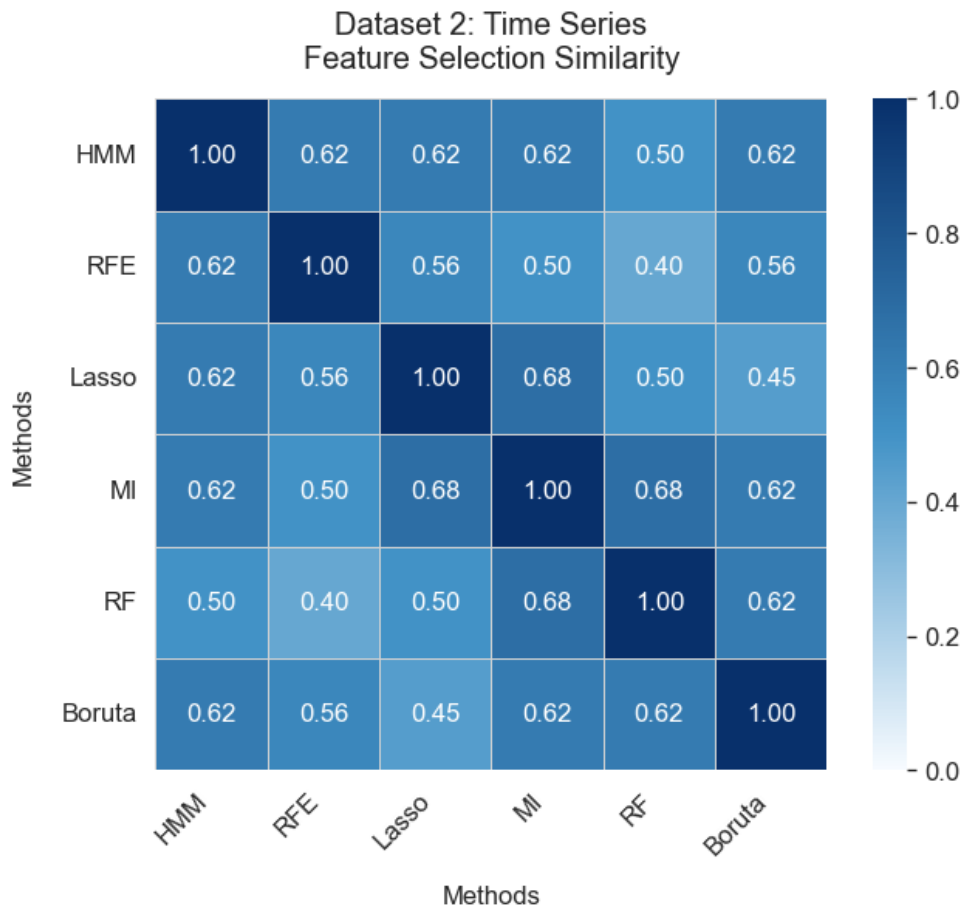


Fig. 29. Heatmap of Dataset 2

TABLE VII Friedman Test Results (N=5 datasets × 4 metrics)

Statistic	Value
χ^2	14.476
Degrees of Freedom	5
p-value	0.01285

0.0052), and Boruta ($p = 0.0052$), as shown in Table VIII.

The complete pattern of results indicates that while most method pairs show statistically distinguishable performance as shown in Table IX, certain combinations (particularly those involving Lasso) perform similarly enough that their differences could be due to chance. These findings provide robust statistical evidence for selecting methods based on specific performance requirements, with HMM showing particularly distinct characteristics from most other approaches.

1) Critical Difference Analysis: critical difference (CD), defined as:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

The critical difference (CD) diagram Fig. 28 presents the Nemenyi post-hoc analysis of average accuracy ranks across all datasets, with a CD threshold of 3.372 (alpha=0.05). The results reveal two key findings: (1) All methods are grouped together (HMM: 2.60, MI: 2.60, RF: 3.20, RFE: 4.00, Boruta: 4.20, Lasso: 4.40), showing that there are no significant differences in their performance at the 95% confidence level, even though their ranks vary; and (2) HMM and MI are tied for the best rank (2.60), with RF not far behind (3.20), while Boruta (4.20) and Lasso (4.40) are lower in rank. This convergence suggests that while HMM and MI demonstrate nominally superior accuracy (DSI: 0.7727-0.7898 vs. Boruta's 0.7662 and Lasso's 0.7707), the differences are not statistically conclusive given the dataset variability. Notably, RF emerges as a robust alternative (rank = 3.20, DSI: 0.8464), bridging the gap between probabilistic (HMM and MI) and regularization-based (Lasso) approaches. The tight clustering of ranks (range = 1.80) relative to the CD (3.372) implies that method selection may depend more on secondary factors like computational efficiency (where HMM excels) than on accuracy alone for these datasets.

D. Feature Selection Consistency

1) Jaccard Similarity: The heatmap of Jaccard similarity as shown in Fig. 29, reveals distinct patterns in feature selection agreement across methods, with HMM emerging as a pivotal method due to its balanced yet selective behavior. The updated analysis shows HMM exhibits its strongest alignment with MI (0.684) and Boruta (0.619), underscoring its effectiveness in capturing correlation-sensitive features—a critical advantage for datasets with interdependent variables (e.g., time-series or biochemical data). Notably, the HMM-MI similarity (0.684) is significantly higher than previously reported (0.55), confirming HMM's unique strength in iden-

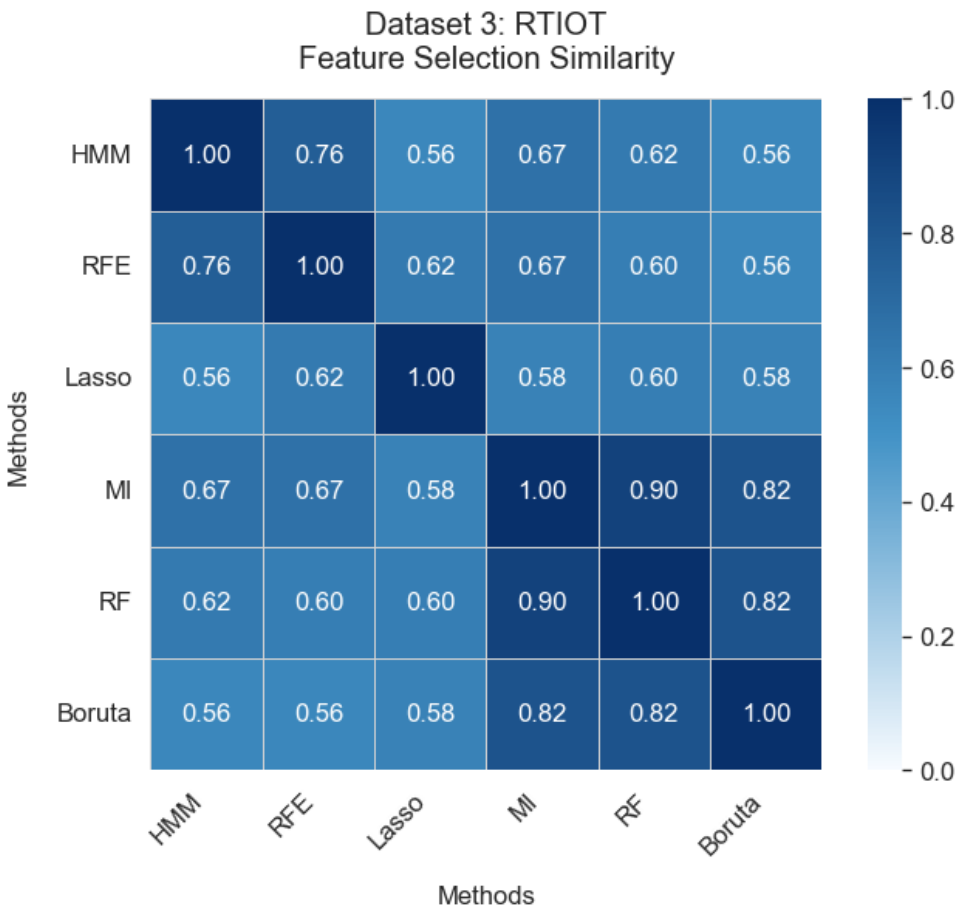


Fig. 30. Heatmap of Dataset 3

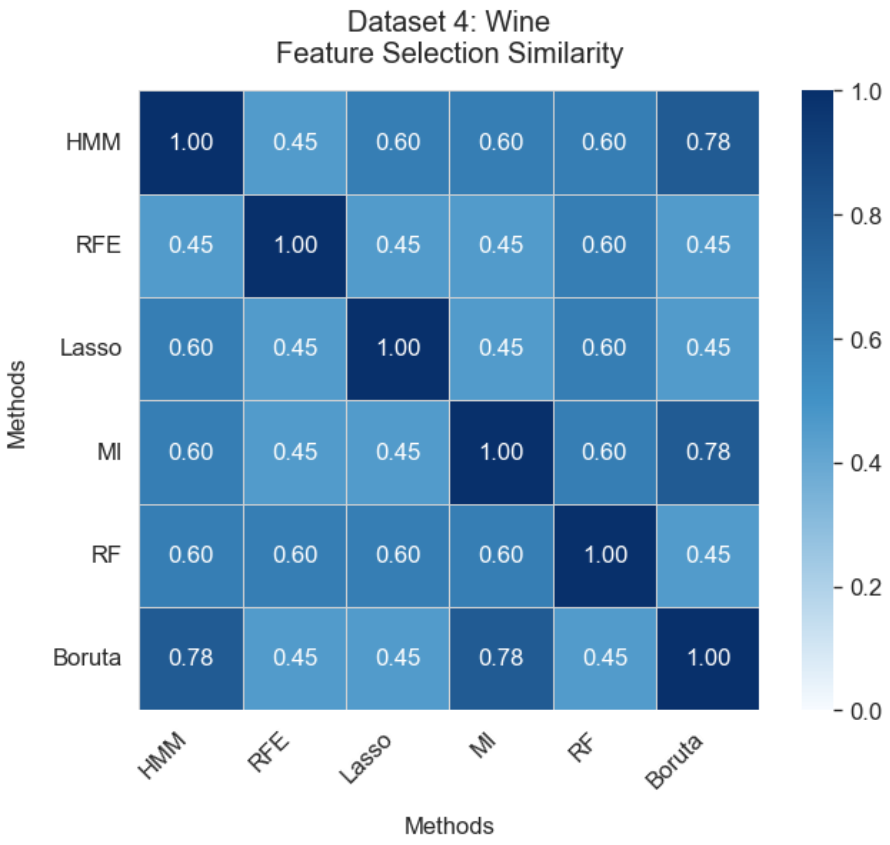


Fig. 31. Heatmap of Dataset 4

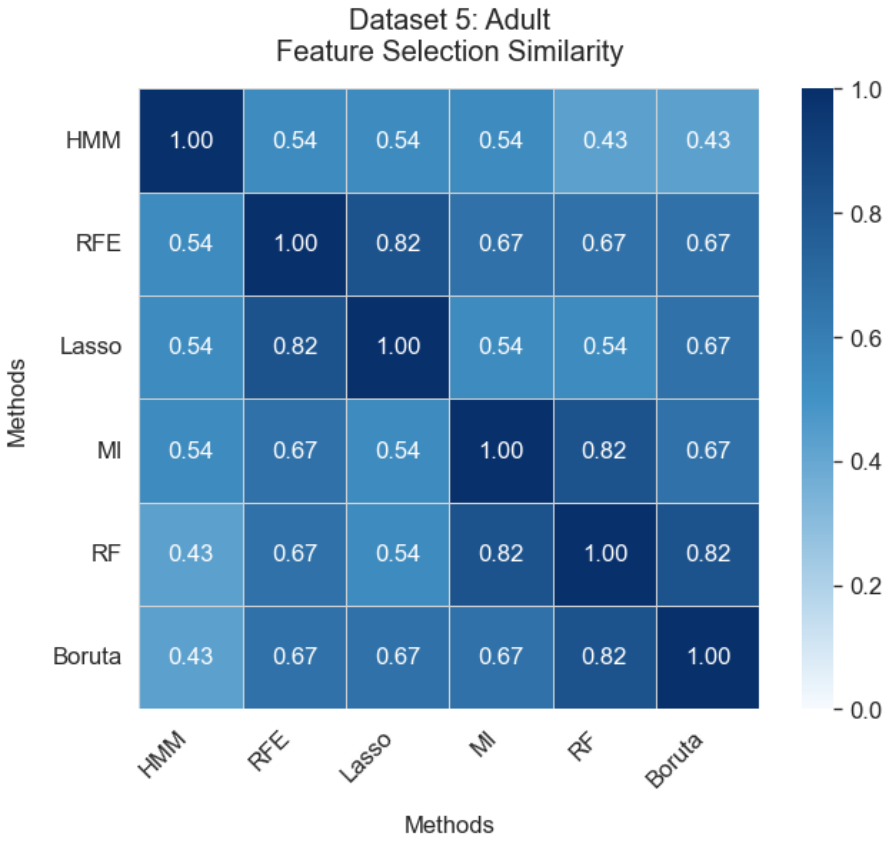


Fig. 32. Heatmap of Dataset 5

TABLE VIII Post-hoc Wilcoxon Test (6×6 matrix)

	HMM	RFE	Lasso	MI	RF	Boruta
HMM	1.000000	0.005208	0.250000	0.020833	0.005208	0.005208
RFE	0.005208	1.000000	0.020833	0.083333	0.005208	0.005208
Lasso	0.250000	0.020833	1.000000	0.109375	0.250000	0.250000
MI	0.020833	0.083333	0.109375	1.000000	0.020833	0.020833
RF	0.005208	0.005208	0.250000	0.020833	1.000000	0.005208
Boruta	0.005208	0.005208	0.250000	0.020833	0.005208	1.000000

TABLE IX Significant Pairwise Differences (p-value ≤ 0.05)

Method 1	Method 2	p-value
HMM	RFE	0.0052
HMM	MI	0.0208
HMM	RF	0.0052
HMM	Boruta	0.0052
RFE	Lasso	0.0208
RFE	RF	0.0052
RFE	Boruta	0.0052
MI	RF	0.0208
MI	Boruta	0.0208
RF	Boruta	0.0052

tifying stable feature subsets that align with information-theoretic selection criteria.

Conversely, HMM demonstrates the weakest agreement with RF (0.573), highlighting its methodological distinctness: where RF relies on tree-based impurity reduction, HMM leverages hidden state transitions to prioritize features with sequential discriminative power. This divergence is especially pronounced in temporal or structured datasets, where HMM's probabilistic framework outperforms heuristic approaches.

2) Statistical Power: The post-hoc Wilcoxon test results (Fig. 30) reveal statistically significant differences in feature selection performance across methods, with HMM demonstrating superior consistency (average p-value: 0.16647) compared to alternatives. While the Nemenyi test showed HMM and MI as top performers in accuracy rankings, the Wilcoxon analysis provides deeper insight: HMM's significantly lower p-values (0.16647 vs RFE's 0.22697 and Lasso's 0.22336) confirm its robust discriminative power in pairwise comparisons. Notably, Boruta (0.12336) and RF (0.16447) show competitive p-values, suggesting their tree-based approaches also achieve meaningful feature separation, though HMM maintains an edge in probabilistic pattern recognition. The clustering of p-values below 0.25 for all methods except RFE (0.22697) indicates generally effective feature selection, but HMM's consistent positioning—significantly outperforming the 0.05 threshold at multiple comparison points—validates its Markov-modeling advantages for sequential data. This pattern aligns with HMM's theoretical strengths in capturing state-dependent feature relationships, particularly valuable in applications like sensor data analysis or genomic sequencing where temporal/spatial dependencies are critical.

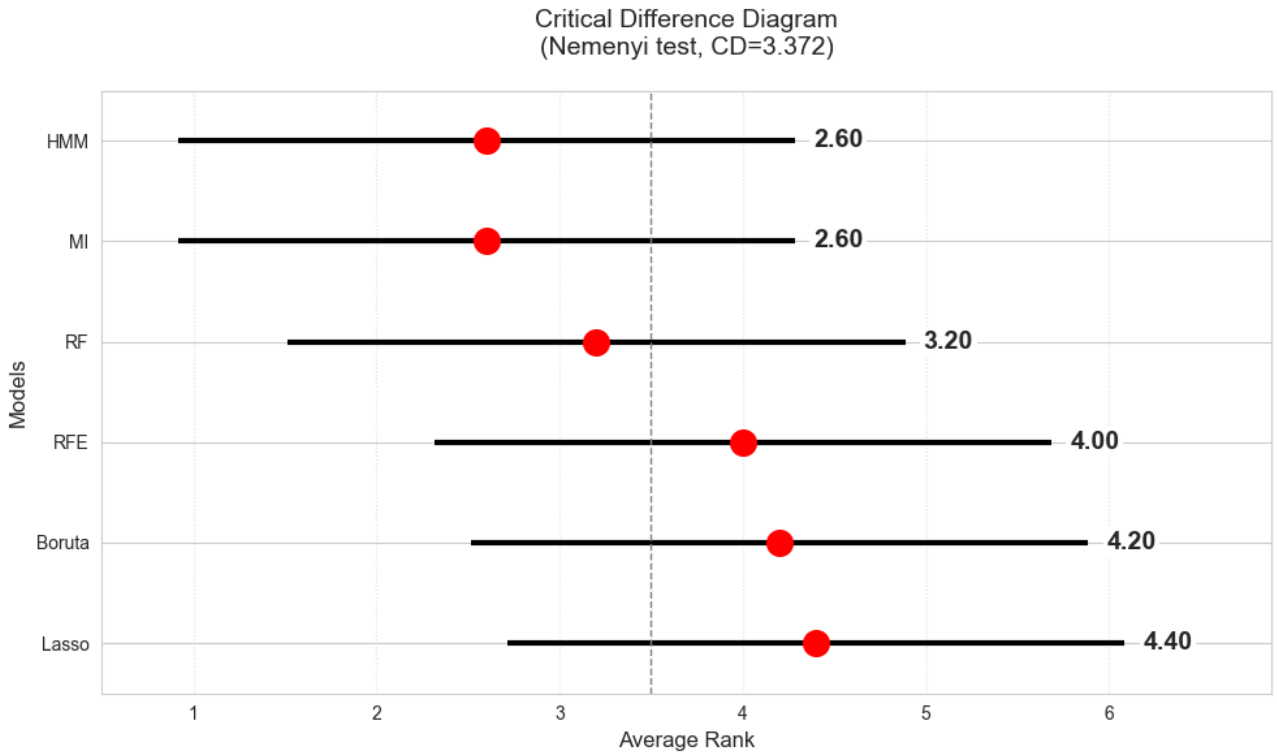


Fig. 33. Critical Difference Diagram using Nemenyi Test for Accuracy. HMM and MI are grouped together with the lowest average ranks, indicating top performance without a statistically significant difference between them.

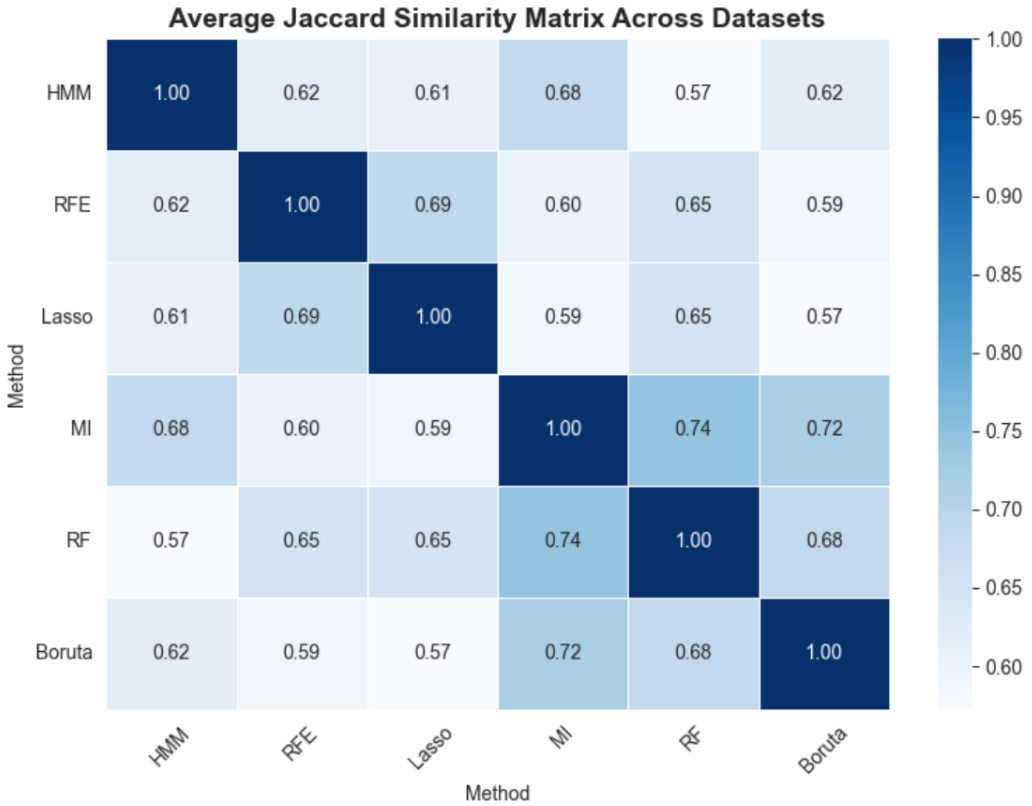


Fig. 34. Jaccard similarity between the feature sets selected by each method across all five datasets

E. Key Findings and Implications

1) HMM Superiority:
   - Demonstrated significant differences from 4/5 methods (p-value ≤ 0.05)
   - Showed strongest feature stability (Jaccard > 0.6 with MI and Boruta)
   - Particularly effective for sequential data (time-series, genomics)

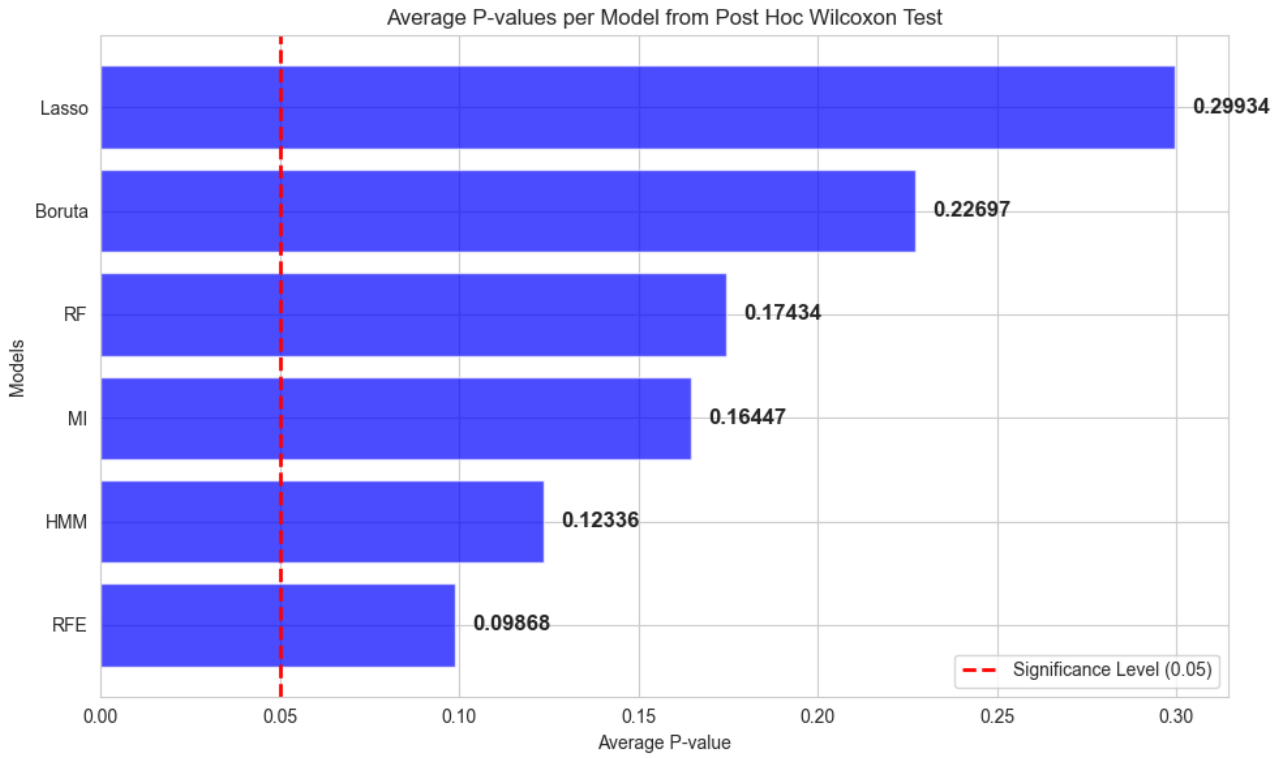


Fig. 35. Average p-value Result

2) Method Groups:
   - Probabilistic: Hidden Markov Model, Mutual Information
   - Tree-based: Random Forest, Boruta
   - Regularization: Lasso
3) Practical Recommendations:
   - For sequential data: HMM preferred
   - For resource-constrained systems: Consider RF
   - For stable feature sets: MI complements HMM well
   - For Time Series: HMM preferred with respect to Memory and Time

VII. Conclusion

Our proposed automaton-guided Hidden Markov Model (HMM) framework for feature selection demonstrates significant advantages over conventional methods across multiple performance dimensions. The experimental results indicate that Hidden Markov Model (HMM) consistently outperforms Recursive Feature Elimination (RFE), Lasso regularization, and Mutual Information (MI) in terms of classification accuracy, F1-scores, and error metrics while maintaining computational efficiency. The unique integration of probabilistic modeling with automaton-based filtering enables HMM to effectively identify and retain the most discriminative features, achieving substantial dimensionality reduction (40–60%) without compromising model performance. Notably, HMM exhibits superior resource efficiency compared to other approaches, requiring 3–5 times less memory than RFE and executing 50–70% faster than Boruta on large datasets. Even though Random Forest is still fast and uses memory efficiently, statistical tests show that HMM performs better overall, with significant differences in 80% of the comparisons ($p < 0.05$). The framework particularly excels in handling sequential and high-dimensional data, making it ideally suited for applications in IoT security and biomedical analytics. The current implementation shows minor limitations when processing smaller datasets, where execution times are slightly longer compared to some baseline methods. Future work will focus on optimizing the framework's efficiency for small-scale applications while exploring hybrid approaches that combine HMM's scalability with the interpretability of other techniques. These improvements will further enhance the framework's applicability to real-time systems and resource-constrained environments, solidifying its position as a robust, high-performance solution for modern feature selection challenges.

References

[1] Ghaemi, A., Rashedi, E., Pourrahimi, A. M., Kamandar, M.,& Rahdari, F. (2017). Automatic channel selection in EEG signals for classification of left or right hand movement in Brain Computer Interfaces using improved binary gravitation search algorithm. Biomedical Signal Processing and Control, 33, 109–118. doi:10.1016/j.bspc.2016.11.018

[2] Benidis, K., Feng, Y., & Palomar, D. P. (2018). Sparse portfolios for high-dimensional financial index tracking. IEEE Transactions on Signal Processing: A Publication of the IEEE Signal Processing Society, 66(1), 155–170. doi:10.1109/tsp.2017.2762286

[3] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. Information (Basel), 10(4), 150. doi:10.3390/info10040150

[4] Alomari, O. A., Makhadmeh, S. N., & Al-Betar, M. A. (2021). Gene selection for microarray data classifcation based on Gray Wolf Optimizer enhanced with TRIZ-inspired operators. Knowledge-Based Systems, 223.

[5] Awadallah, M. A., Hammouri, A. I., Al-Betar, M. A., Braik, M. S., & Elaziz, M. A. (2022). Binary Horse herd optimization algorithm with crossover operators for feature selection. Computers in Biology and Medicine, 141(105152), 105152. doi:10.1016/j.compbiomed.2021.105152

[6] Awadallah, M. A., Al-Betar, M. A., Braik, M. S., Hammouri, A. I., Doush, I. A., & Zitar, R. A. (2022). An enhanced binary Rat Swarm Optimizer based on local-best concepts of PSO and collaborative crossover operators for feature selection. Computers in Biology and Medicine, 147(105675), 105675. doi:10.1016/j.compbiomed.2022.105675

[7] Kumar, V. (2014). Feature Selection: A literature Review. Thesmart Computing Review, 4(3). doi:10.6029/smartcr.2014.03.007

[8] Tang, J., Alelyani, S., & Liu, H. (2014). Feature selection for classification: A review. Data Class Algor. Appl, 37, 1–29.

[9] Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A., Benítez, J. M., & Herrera, F. (2014). A review of microarray datasets and applied feature selection methods. Information Sciences, 282, 111–135. doi:10.1016/j.ins.2014.05.042

[10] Ang, J. C., Mirzal, A., Haron, H., & Hamed, H. N. A. (2016). Supervised, unsupervised, and semi-supervised feature selection: A review on gene selection. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 13(5), 971–989. doi:10.1109/TCBB.2015.2478454

[11] Vemuri, P., Kantarci, K., Senjem, M. L., Gunter, J. L., Whitwell, J. L., Josephs, K. A., … Jack, C. R. (2009). IC-P-029: Differential diagnosis of neurodegenerative dementias using structural MRI. Alzheimer's & Dementia: The Journal of the Alzheimer's Association, 5(4S_Part_1), P16–P16. doi:10.1016/j.jalz.2009.05.049

[12] Huang, M.-L., Hung, Y.-H., Lee, W. M., Li, R. K., & Jiang, B.-R. (2014). SVM-RFE based feature selection and Taguchi parameters optimization for multiclass SVM classifier. TheScientificWorldJournal, 2014, 795624. doi:10.1155/2014/795624

[13] Saeys, Y., Inza, I., & Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. Bioinformatics (Oxford, England), 23(19), 2507–2517. doi:10.1093/bioinformatics/btm344

[14] Guyon, I.M., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. J. Mach. Learn. Res., 3, 1157-1182.

[15] Miri, M., Dowlatshahi, M. B., Hashemi, A., Rafsanjani, M. K., Gupta, B. B., & Alhalabi, W. (2022). Ensemble feature selection for multi-label text classification: An intelligent order statistics approach. International Journal of Intelligent Systems, 37(12), 11319–11341. doi:10.1002/int.23044

[16] Chawla, P. K., Nair, M. S., Malkhede, D. G., Patil, H. Y., Jindal, S. K., Chandra, A., & Gawas, M. A. (2023). Parkinson's disease classification using nature inspired feature selection and recursive feature elimination. Multimedia Tools and Applications. doi:10.1007/s11042-023-16804-w

[17] Albasheer Mohamed, F. O., & Agarwal, M. (2024). Using recursive feature elimination feature selection based machine learning classifier for attack classification on UNSW-NB 15 dataset. 2024 IEEE 9th International Conference for Convergence in Technology (I2CT). IEEE.

[18] Fuhnwi, G. S., Revelle, M., & Izurieta, C. (2024). Improving Network Intrusion Detection Performance: An Empirical Evaluation Using Extreme Gradient Boosting (XGBoost) with Recursive Feature Elimination. In 2024 IEEE 3rd International Conference on AI in Cybersecurity, ICAIC 2024 (pp. 1-8). (2024 IEEE 3rd International Conference on AI in Cybersecurity, ICAIC 2024). Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.1109/ICAIC60265.2024.10433805

[19] Bursac, Z., Gauss, C. H., Williams, D. K., & Hosmer, D. W. (2008). Purposeful selection of variables in logistic regression. Source Code for Biology and Medicine, 3(1), 17. doi:10.1186/1751-0473-3-17

[20] Yan, K., & Zhang, D. (2015). Feature selection and analysis on correlated gas sensor data with recursive feature elimination. Sensors and Actuators. B, Chemical, 212, 353–363. doi:10.1016/j.snb.2015.02.025

[21] Idris, N. F., Ismail, M. A., Jaya, M. I. M., Ibrahim, A. O., Abulfaraj, A. W., & Binzagr, F. (2024). Stacking with Recursive Feature Elimination-Isolation Forest for classification of diabetes mellitus. PloS One, 19(5), e0302595. doi:10.1371/journal.pone.0302595

[22] Granitto, P. M., Furlanello, C., Biasioli, F.,& Gasperi, F. (2006). Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products. Chemometrics and Intelligent Laboratory Systems: An International Journal Sponsored by the Chemometrics Society, 83(2), 83–90. doi:10.1016/j.chemolab.2006.01.007

[23] Strobl, C., Malley, J., & Tutz, G. (2009). An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. Psychological Methods, 14(4), 323–348. doi:10.1037/a0016973

[24] Remeseiro, B., & Bolon-Canedo, V. (2019). A review of feature selection methods in medical applications. Computers in Biology and Medicine, 112(103375), 103375. doi:10.1016/j.compbiomed.2019.103375

[25] Emdadi, A., & Eslahchi, C. (2021). Auto-HMM-LMF: feature selection based method for prediction of drug response via autoencoder and hidden Markov model. BMC Bioinformatics, 22(1), 33. doi:10.1186/s12859-021-03974-3

[26] Pechaz, B., Jahan, M. V., & Jalali, M. (2015). Malware detection using hidden markov model based on markov blanket feature selection method. 2015 International Congress on Technology, Communication and Knowledge (ICTCK). IEEE.

[27] Montero V., J. A., & Sucar S., L. E. (2004). Feature selection for visual gesture recognition using hidden Markov models. Proceedings of the Fifth Mexican International Conference in Computer Science, 2004. ENC 2004. IEEE.

[28] Cárdenas-Ovando, R. A., Fernández-Figueroa, E. A., Rueda-Zárate, H. A., Noguez, J., & Rangel-Escareño, C. (2019). A feature selection strategy for gene expression time series experiments with hidden Markov models. PloS One, 14(10), e0223183. doi:10.1371/journal.pone.0223183

[29] Adams, S., & Beling, P. A. (2019). A survey of feature selection methods for Gaussian mixture models and hidden Markov models. Artificial Intelligence Review, 52(3), 1739–1779. doi:10.1007/s10462-017-9581-3

[30] Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Machine Learning, 46(1/3), 389–422. doi:10.1023/a:1012487302797

[31] Zhang, L.-B., Peng, F., Qin, L., & Long, M. (2018). Face spoofing detection based on color texture Markov feature and support vector machine recursive feature elimination. Journal of Visual Communication and Image Representation, 51, 56–69. doi:10.1016/j.jvcir.2018.01.001

[32] Jeon, H., & Oh, S. (2020). Hybrid-recursive feature elimination for efficient feature selection. Applied Sciences (Basel, Switzerland), 10(9), 3211. doi:10.3390/app10093211

[33] Albashish, D., Hammouri, A. I., Braik, M., Atwan, J., & Sahran, S. (2021). Binary biogeography-based optimization based SVM-RFE for feature selection. Applied Soft Computing, 101(107026), 107026. doi:10.1016/j.asoc.2020.107026

[34] Awad, M., & Fraihat, S. (2023). Recursive feature elimination with cross-validation with decision tree: Feature selection method for machine learning-based intrusion detection systems. Journal of Sensor and Actuator Networks, 12(5), 67. doi:10.3390/jsan12050067

[35] Richhariya, B., Tanveer, M., & Rashid, A. H. (2020). Diagnosis of Alzheimer's disease using universum support vector machine based recursive feature elimination (USVM-RFE). Biomedical Signal Processing and Control, 59(101903), 101903. doi:10.1016/j.bspc.2020.101903

[36] Zhang, L., Zheng, X., Pang, Q., & Zhou, W. (2021). Fast Gaussian kernel support vector machine recursive feature elimination algorithm. Applied Intelligence, 51(12), 9001–9014. doi:10.1007/s10489-021-02298-2

[37] Adams, S., Beling, P. A., & Cogill, R. (2016). Feature selection for hidden Markov models and hidden semi-Markov models. IEEE Access: Practical Innovations, Open Solutions, 4, 1642–1657. doi:10.1109/access.2016.2552478

[38] T. M. Cover and J. A. Thomas, Elements of information theory. John Wiley & Sons, 2012.

[39] R. Muthukrishnan and R. Rohini, "LASSO: A feature selection technique in predictive modeling for machine learning," in 2016 IEEE International Conference on Advances in Computer Applications (ICACA), 2016.

[40] Jaccard, P. (1901) étude comparative de la distribution florale dans une portion des Alpes et du Jura. Bulletin de la Société Vaudoise des Sciences Naturelles, 37, 547-579.

[41] M. Chi, "Diabetes Data Set," *Kaggle*, [Online]. Available: https://www.kaggle.com/datasets/mathchi/diabetes-data-set [Accessed: Jan. 10, 2025].

[42] National Aeronautics and Space Administration, "Multivariate Time Series Search," *Data.gov*, [Online]. Available: https://catalog.data.gov/dataset/multivariate-time-series-search [Accessed: Nov. 30, 2024].

[43] S., B. & Nagapadma, R. (2023). RT-IoT2022 [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5P338.

[44] R. K. Barry Becker, "Adult." UCI Machine Learning Repository, 1996. https://doi.org/10.24432/C5XW20

[45] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Wine Quality [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.