Electrical Load Forecasting Using a Hybrid Large Margin Nearest Neighbor Method

Alieh Ashoorzadeh [1] , Abbas Toloie Eshlaghy•[2] and Mohammad Ali Afshar Kazemi[3]

***Abstract*--Load forecasting is a key component of electric utility operations and planning. Because of today's highly developed electricity markets and rapidly growing power systems, load forecasting is becoming an essential part of power system operation scheduling. This paper proposes a new short-term load forecasting model based on the large margin nearest neighbor (LMNN) classification algorithm to improve prediction accuracy. The accuracy of many classification methods, such as k-nearest neighbor (k-NN), is significantly influenced by the technique used to calculate sample distances. The Mahalanobis distance is one of the most widely used methods for calculating distance. Numerous techniques have been used to enhance k-NN performance in recent years, including LMNN. Our proposed approach aims to solve the local optimum problem of LMNN, compute data similarities, and optimize the cost function that establishes the distances between instances. Before using gradient descent to determine the ideal parameter values for the cost function, we employ a genetic algorithm to shrink the size of the solution space. Additionally, our method's forecasting errors are contrasted with those of the BPNN and ARMA models. The comparative findings show how well the recommended forecasting model performs in short-term load forecasting.**

***Index Terms*-- Short-Term Load Forecasting; Large Margin Nearest Neighbor; Distance learning; Genetic Algorithm.**

I. INTRODUCTION

To achieve specific accuracy requirements, power system load forecasting refers to the study of or use of a mathematical method to systematically process past and future loads, accounting for significant system operational features, capacity expansion decisions, environmental factors, and social implications. Improving load forecasting techniques helps with planned power management, which helps with building reasonable power supply construction plans, facilitating power improvement, and maximizing the system's economic and social benefits. It also helps rationally organize the grid operation mode and unit maintenance plan.

With more accuracy than long-term load forecasting, short-term load forecasting is used to predict the power loads in the upcoming months, weeks, or even days. Forecasting accuracy is crucial in power demand management because it directly affects operators' economic costs in the competitive power market [1]. Short-term load forecasting data indicate that not only can it optimize the combination of generator sets, economical dispatching, and the calculation of power flow for power generation, but it also ensures the financially secure operation of the power system [2].

Short-term load forecasting (STLF) is primarily conducted using methods such as convolutional neural networks [3], fuzzy time series [4], and genetic algorithms [5], among other methods. Despite the widespread use of these techniques, several issues remain. For instance, (1) forecasting is complicated and simple mathematical formulas are not sufficient enough to solve this problem; (2) external factors such as weather conditions and consumer demands can lead to a dynamic environment that makes load forecasting very challenging; and (3) often models overfit or fall into local optima. Therefore, it is crucial to develop more precise and easier-to-understand models.

Short-term load forecasting is a crucial aspect of power system operations, enabling grid operators to make informed decisions regarding generation scheduling, demand response, and system reliability. With the growing integration of renewable energy sources and smart grid technologies, the accuracy and efficiency of STLF have become increasingly important. Over the past decade, significant advancements have been made in this field, driven by developing novel forecasting models, integrating big data analytics, and applying machine learning techniques.

The *k*-nearest neighbor (k-NN) algorithm [6] is a helpful tool that can be simply implemented for forecasting. It should, therefore, be one of the first options when there is little or no prior knowledge about the distribution data because it is frequently used to solve nonlinear problems in which the collected data do not always follow the linear assumption. Furthermore, it effectively minimises the effects of the variables on the experimental processes [7]. It makes no assumptions about the collected data; however, it is sensitive to outliers.

To capture uncertainty and reflect the range of electrical load fluctuation, Dong et al. [8] proposed a deep learning strategy based on *k*-NN to solve the high computational cost due to the intricate network structure. The k-NN's approach is first used to find elements of previous electrical load time series similar to the future values by measuring the distance between the training and testing datasets. Then, for multi-objective optimization, the second generation of the non-dominated sorting genetic algorithm is used to determine the maximum forecasting accuracy and the smallest category number of k-nearest neighbors. The prediction intervals are obtained using modified non-parameter kernel density estimation based on the network's forecasting outcomes.

To implement LMNN for short-term load forecasting, the

1. Department of Information Technology Management, Science and Research Branch, Islamic Azad University, Tehran, Iran.
2. Department of Industrial Management, Science and Research Branch, Islamic Azad University, Tehran, Iran.
3. Department of Industrial Management, Central Tehran Branch, Islamic Azad University, Tehran, Iran.
• Corresponding author Email: toloie@gmail.com

algorithm must first be trained on historical load data to identify the underlying patterns and correlations. The model can then predict the short-term load demand using the present input attributes. With the help of this strategy, utilities will be able to decide with confidence on resource allocation, energy production, and grid stability shortly.

Like every forecasting technique, LMNN is not without its pitfalls. The dynamic nature of power consumption patterns, which are impacted by several variables, including the holidays and unforeseen events, is one major difficulty. The model must be modified for accurate forecasts to account for these differences. Furthermore, there may be issues with data availability and quality because erroneous or missing data could make the model unreliable [9].

LMNN must be continuously improved to overcome the obstacles and increase accuracy and reliability. This research aims to improve the algorithm's ability to adjust to changing load patterns by adding sophisticated features considering other factors. Another important area of research is the inclusion of real-time data streams, which makes forecasts more resilient to unanticipated occurrences by enabling the model to react quickly to abrupt changes in demand.

LMNN models can generate more reliable forecasts if the input data is precise, comprehensive, and indicative of the real load conditions. The production of comprehensive datasets that represent a variety of scenarios is made possible by cooperative efforts between utilities, research institutes, and data suppliers. This allows the algorithm to generalize effectively across a range of settings [10].

Deploying LMNN has broader goals than just improving operations right away. LMNN's precise forecasting is becoming increasingly important for utilities as they work to incorporate renewable energy sources into the grid and balance supply and demand. This is consistent with the larger goal of decreasing environmental impact and fostering sustainability. Through joint research, the energy industry may overcome obstacles and improve its capabilities, paving the way for more adaptable and robust short-term load forecasting, ultimately contributing to a more sustainable and efficient energy ecosystem.

A. Motivation and Contribution

Load forecasting plays a significant role in the planning and operations of electric utilities. Because of the highly developed electricity markets and the rapidly growing power systems of the modern world, load forecasting is becoming an essential part of power system operation scheduling. If the load forecasting is accurate, there is a good possibility of savings in control operations and decision-making, such as dispatch, unit commitment, fuel allocation, power system security assessment, and off-line analysis. Consequently, improving the accuracy of short-term load forecasting has always been the primary objective of load forecasting research.

Our proposed approach seeks to solve the local optimum problem of LMNN, optimizes the cost function that determines the distances between instances, and introduces a cost function to calculate the fitness value. Due to the issues with the *k*-NN and LMNN methods, we first use the genetic algorithm to narrow down the range of the solution space. Then, we use gradient descent to determine the optimal value of the parameter in the cost function. This allows us to optimize the objective function in our method to obtain the distance for the test data and more accurate results.

The main contribution of this paper lies in developing a novel short-term load forecasting model that combines the Large Margin Nearest Neighbor (LMNN) algorithm with a hybrid optimization approach using genetic algorithms and gradient descent. Key contributions include: (1) Proposing a method that addresses limitations in traditional LMNN and k-NN algorithms, such as sensitivity to local optima and distance metric inefficiencies. (2) Incorporation of distance learning in a way that the method optimizes the Mahalanobis distance metric within LMNN to better classify and forecast electricity loads, enhancing the algorithm's adaptability to dynamic energy consumption patterns. (3) Improving forecasting accuracy and demonstrating superior performance in short-term load forecasting compared to traditional methods like Autoregressive Moving Average (ARMA) and Back-Propagation Neural Network (BPNN), as evidenced by lower forecasting errors (e.g., RMSE, NMSE). (4) Validating the forecasting model using real-world hourly electricity load data from the National Electricity Market of Australia, showing its utility in modern power systems.

The rest of this paper is structured as follows: a literature review is in Section II. Materials and the proposed method are discussed in Section III. Simulation results are shown and discussed in Section IV. Finally, the conclusion is presented in Section V.

II. LITREATURE REVIEW

Ashfaq and Javaid [11] addressed the problem of forecasting electricity prices and loads by introducing an improved new technique. Their upgraded technique framework includes feature engineering and classification. Feature selection and feature extraction are components of feature engineering. For feature selection, Decision Tree Regression (DTR) is employed. Redundancy in features is removed through feature selection using Recursive Feature Elimination (RFE). Singular Value Decomposition (SVD) is used in feature extraction, the second step of feature engineering, to minimize the dimensionality of features. Forecasting and load prediction are the final steps. Two current methods, *k*-NN and Multi-Layer Perceptron (MLP), along with a new method called Enhanced *k*-NN (EKNN), were utilized to forecast power load and pricing. The accuracy of the suggested technique is superior to that of MLP and k-NN.

The approach for Time Series Forecasting (TSF) based on the *k*-Nearest Features in Time Series (KNFTS) and *k*-Nearest Patterns in Time Series (KNPTS) algorithms, two variations of the *k*-NN method was suggested by Gómez-Omella et al. [12]. These algorithms are used to identify comparable electricity usage patterns and then provide future forecasts, while only a historical data set comprising the time and energy consumption variables is used. Additionally, it appears that using elastic similarity metrics like Dynamic Time Warping (DTW) and Edit Distance for Real Sequences (EDR) might be preferable to other error measurements.

A study by Marino, Amarasinghe, and Manic [13] introduced a long short-term memory (LSTM) network for STLF, demonstrating that LSTM outperforms traditional neural networks by effectively capturing temporal dependencies in load data. Their work highlighted the importance of memory mechanisms in forecasting models, particularly in handling non-linear and non-stationary time series data. Similarly, Ryu, Noh, and Kim [14] compared

various deep learning architectures, including convolutional neural networks (CNNs) and gated recurrent units (GRUs), concluding that hybrid models combining CNN and LSTM layers provide superior forecasting performance due to their ability to extract both spatial and temporal features from load data.

Integrating external factors, such as weather conditions and social events, into ML models has also been a focal point in recent research. Kong et al. [15] proposed a hybrid model that combines LSTM with attention mechanisms to focus on relevant features in the input data selectively. This approach significantly improves forecasting accuracy by dynamically adjusting the importance of different inputs based on their relevance to the prediction task. The study underscores the potential of attention-based models in enhancing the interpretability and performance of STLF models.

Ensemble methods, which combine multiple models to improve forecasting accuracy, have seen considerable adoption in STLF. These methods leverage the strengths of individual models while mitigating their weaknesses, leading to more robust predictions. For instance, Khwaja et al. [16] proposed an ensemble framework that integrates support vector regression (SVR), random forests (RF), and deep learning models. Their approach demonstrated that ensemble models consistently outperform single models across various performance metrics, including mean absolute percentage error (MAPE) and root mean square error (RMSE).

Another significant trend is the development of hybrid models that combine different forecasting techniques to capture diverse characteristics of load data. Bashir et al. [17] introduced a hybrid model that integrates wavelet transform, LSTM, and a time-varying seasonal model. Their study showed that the hybrid model outperforms standalone models by effectively decomposing the load data into different frequency components, allowing each component to be forecasted using the most suitable technique. This research highlights the potential of hybrid models in addressing the challenges posed by the complex and dynamic nature of load data.

The proliferation of smart meters and the increasing availability of high-frequency data have opened new avenues for improving STLF. The use of big data analytics in STLF has been explored by several researchers, focusing on integrating large-scale datasets and developing real-time forecasting models.

In a study by Li et al. [18], the authors utilized big data analytics to process and analyze vast amounts of load and weather data, demonstrating that real-time data integration significantly enhances the accuracy of STLF. The study employed a big data platform based on Apache Spark to handle the computational demands of processing high-frequency data, showcasing the feasibility of real-time load forecasting in modern power systems. This research emphasizes the importance of scalable and efficient data processing frameworks in the era of big data.

Moreover, the role of feature selection and dimensionality reduction techniques in handling large datasets has been a key focus. Bezerra et al. [19] proposed a feature selection method based on mutual information and principal component analysis (PCA) to reduce the dimensionality of input data while retaining the most informative features. Their approach demonstrated that carefully selecting and transforming input features leads to more accurate and computationally efficient forecasting models.

Subbiah and Chinnappan [20] proposed RMR-HFS-LSTM, a deep learning model combining Long Short-Term Memory (LSTM) with hybrid feature selection, to improve short-term load forecasting accuracy. Integrating filter (RReliefF, mutual information) and wrapper (RFE) methods reduces dimensionality and overfitting. Experiments on European electricity data show that RMR-HFS-LSTM outperforms MLP and RNN in MAPE and RMSE metrics.

Neeraj et al. [21] introduce the Singular Spectrum Analysis-Long Short-Term Memory (SSA-LSTM) model for electrical load forecasting, leveraging signal processing to address the challenges of noisy and irregular data. SSA, a signal processing technique, is used to filter out noise from skewed load series, and the processed data is then used by the LSTM model for accurate forecasting. Evaluated using five datasets from the Australian Energy Market Operator (AEMO), SSA-LSTM outperforms several state-of-the-art models, including persistence, AR, ARMAX, SVR, RF, ANN, DBN, and others, in terms of RMSE and MAPE for both half-hourly and one-day ahead load forecasting.

Despite the advancements in STLF, several challenges remain. One of the main challenges is handling uncertainty and variability in load data, particularly with the increasing penetration of renewable energy sources. To address this, recent studies have explored probabilistic forecasting methods that provide a range of possible outcomes rather than a single-point estimate. For example, Jensen et al. [22] proposed a probabilistic forecasting framework that combines quantile regression with a deep learning model to generate prediction intervals. This approach allows grid operators to assess the uncertainty associated with load forecasts and make more informed decisions.

Another challenge is the interpretability of complex machine learning models. As models become more sophisticated, understanding their decision-making process becomes increasingly difficult. Recent research has focused on developing interpretable models that balance accuracy with transparency. For instance, Moon et al. [23] introduced an interpretable neural network model that incorporates explainability techniques such as SHAP (Shapley Additive Explanations) values, enabling stakeholders to understand the contribution of each input feature to the final forecast.

Research gaps that could be handled using a hybrid method include:

Improving prediction accuracy with complex patterns: load forecasting needs to handle complex patterns such as seasonality, holidays, and unusual demand spikes. Simple *k*-NN models may struggle with these intricate patterns, especially if the data is sparse or noisy. LMNN can help by optimizing the distance metric, focusing on finding relevant neighbors even in complex, non-linear relationships. By applying LMNN, the hybrid model can better capture intricate demand patterns and improve forecasting accuracy.

Noise reduction and robustness: load forecasting data can be noisy, with missing values or sensor errors. *k*-NN models may be sensitive to noise, leading to inaccurate predictions. LMNN's ability to learn a robust distance metric can reduce the effect of noisy data by adjusting the space in which *k*-NN operates. The hybrid model can better handle noise and outliers, increasing the robustness of the forecasting model.

Handling non-linear relationships: load forecasting often involves non-linear relationships between input features (e.g. time of day and consumer behavior). *k*-NN is typically better suited for linear relationships and can struggle with non-

linearities. LMNN can be used to learn a more flexible distance metric that captures non-linear relationships between the features. This hybrid method would allow the *k*-NN algorithm to better handle complex, non-linear dependencies in the data.

Adaptability to dynamic and evolving data: Load patterns can evolve due to changes in consumer behavior, economic conditions, or climate. Traditional *k*-NN models can struggle to adapt to such dynamic changes without retraining the entire model. By incorporating LMNN into the hybrid approach, the model can more effectively adapt to changes in the underlying distribution of the data, as LMNN adjusts the distance metric based on new patterns in the data. This allows the hybrid model to adjust and improve over time dynamically.

In summary, short-term load forecasting has seen significant advancements over the past five years, driven by adopting machine learning and deep learning techniques, developing ensemble and hybrid models, and integrating big data analytics. However, challenges such as handling uncertainty, improving model interpretability, and scaling to real-time applications remain active research areas. Continued efforts in these directions are essential for enhancing the reliability and efficiency of STLF in modern power systems.

III MATERIALS AND METHODS

A. The k-NN Algorithm

One popular method for classifying data is the *k*-NN classification algorithm. One of the most fundamental concepts in classification is used by the *k*-NN algorithm [24]. Based on supervised learning, this approach is among the earliest for broad and non-parametric classification. This strategy aims to find the closest *k* data available from the training data. The new instance's distance from the training instance set is first determined. Next, the matching class of this instance is predicted by considering *k* members from the new instance's closest neighbors. The sample size, the choice of distance metric, and the value of *k* are the three main variables that affect the *k*-NN algorithm's performance. The choice of the distance metric significantly impacts the algorithm's accuracy. One of the primary needs of the *k*-NN method is the ability to measure the distance between two data points. The objective of distance metric learning is to derive the distance function (similarity) from the data such that the logically similar data move toward each other while the illogically similar data move away. Many learning algorithms require a metric to calculate the distance or similarity between objects. The distance between objects can be determined using a variety of distance metrics, including the Cosine, Manhattan, and Euclidean distances. However, these metrics are not appropriate for every application, and a more accurate metric can be obtained using the training data [25]. Metric learning techniques have emerged as a result of this. By using training data with concepts and meanings that are comparable to one another, we want to determine the distance function. Different data sets are kept apart from one another. Metric learning techniques, like *k*-NN or *k*-means clustering classification, are typically applied as pre-processing for machine learning and pattern recognition algorithms.

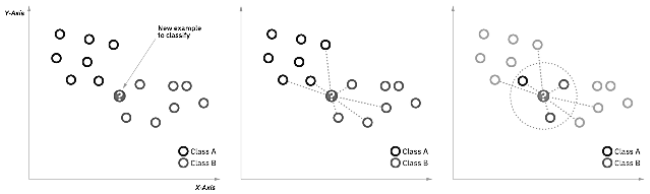


Fig. 1. The k-NN classifier.

The *k*-NN algorithm must determine how far the predicted data point is from the known data point to choose the nearest *k* labeled data, $\{x_1, x_2, \ldots, x_k\}$, where x_1 stands for the known point that is closest to the predicted point; x_2 stands for the known point that is the second closest to the predicted point, and so on. Hence, the *k*-NN algorithm can be used as:

$$s_i = \frac{1}{k} \times \sum_{j=1}^{k} s_{x_j} \tag{1}$$

where s_i is the *i*th predicted value and s_{x_j} is the predicted value of the *j*th closest known point (x_j). The *k*-NN classifier is shown in Fig. 1.

B. Distance Learning Method and LMNN

Researchers have suggested several techniques to obtain distance metric learning over the past few decades [26]. Replacing the Euclidian distance, which does not distinguish between different data features, can significantly increase the accuracy. There are linear methods and non-linear methods for distance metric learning.

The *k*-NN algorithm consists of the following steps:

Step 1: Select the value of *k*.

Step 2: Calculate the distance of *k* number of neighbors.

Step 3: Sort distances in ascending order.

Step 4: Using the calculated distance, select the *k* closest neighbors.

Step 5: Determine how many data items are in each category among these *k* neighbors.

Step 6: Assign to the category where the neighbor count is at its highest.

Assume a set of $X = \{x_i\} \subset R^n$ of data points, the general Mahalanobis distance is as follows:

$$D^2(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^T \mathrm{M} (\vec{x}_i - \vec{x}_j) \tag{2}$$

where M can be any positive matrix that is found by optimization.

Mahalanobis distance is taught using the LMNN metric learning method in the *k*-NN classifier [27]. This learning method places a large distance between dissimilar data and its *k* neighbors who share the same label. A group of *k* nearest neighbors with the same label are considered the target data in the LMNN method for the training data x_i. It is necessary that among the *k* nearest neighbors of the data x_i, all labels are different to carry out a successful *k*-NN classification. As a result, when using the LMNN method, a zone is considered for the data x_i, which includes both the target data and a safety margin. According to this definition, similar data placed in this zone are hard to categorize and are regarded as noisy data. The amount of noisy data in the LMNN learning method needs to be reduced by the learning method.

C. Genetic Algorithm and Gradient Descent

Genetic algorithms [28] are evolutionary algorithms that search and find optimal solutions. It searches a multi-dimensional search space for the best answer. Genetic algorithms produce a set that contains various solutions instead of just one. Finding a suitable solution is more likely when various points are considered. Every single one of them is a vector in the multidimensional space. For problem-

solving, genetic algorithms simulate an evolutionary continuum in a computer environment. Unlike other optimization methods, they develop a set of these structures rather than just one structure for a given solution.

A gene includes each component of an individual. Genetic algorithm processes on an evolutionary continuum to determine the composition of the population as a whole. A population of chromosomes that have been chosen at random typically serves as the starting point for the genetic algorithm. These chromosomes serve as models for potential solutions to the problem. To simulate the natural reproduction and mutation of species, it employs two main operators: crossover and mutation. The fittest chromosomes are favored when choosing which ones to combine and keep alive.

Gradient descent [29] is a general algorithm usually used to find the optimal solution in an unconstrained multivariate differentiable function. Gradient descent is not only used in linear regression but can also be used in many machine learning subjects. In general, this algorithm applies to infinite parameters.

If we start from a point in the function, the fastest way to reach the optimal point is to move along the path with the greatest slope. The gradient of the function, which is the partial derivatives of the function with respect to the variables $\theta_0, \theta_1, \dots, \theta_n$ indicates the greatest slope. Therefore, the formulation of the problem is as follows:

We have a cost function $J(\theta_0, \theta_1, \dots, \theta_n)$ and we want to minimize $\theta_0, \theta_1, \dots, \theta_n$. The algorithm starts with the initial $\theta_0, \theta_1, \dots, \theta_n$. The value of $\theta_0, \theta_1, \dots, \theta_n$ is changed towards better results. The change of $\theta_0, \theta_1, \dots, \theta_n$ is proportional to the partial derivatives of the cost function $(\theta_0, \theta_1, \dots, \theta_n)$.

Changing the value of $\theta_0, \theta_1, \dots, \theta_n$ continues until $J(\theta_0, \theta_1, \dots, \theta_n)\{\theta_i := \theta_i - \propto * \partial/\partial\theta_i \; J(\theta_0, \theta_1, \dots, \theta_n)\}$ reaches the lowest point possible. The final solution may be the local optimum point instead of the global optimum point. In each iteration of the algorithm, the values of $\theta_0, \theta_1, \dots, \theta_n$ are updated simultaneously according to the partial derivatives of the cost function with respect to the parameters. α is called the learning rate and controls the length of steps the algorithm takes in each iteration. Usually, its value is between 0 and 1. If α is chosen to be very small, the convergence happens later because the gradient descent moves towards the minimum point with smaller steps. If α is chosen large, the value of $J(\theta)$ may not decrease with each iteration or it may not reach convergence. Often, the learning rate is set at 0.1 [30].

D. Encoding Scheme

In the context of the STLF method using LMNN and GA, an effective encoding scheme for representing potential solutions is crucial. This encoding determines how candidate solutions are structured and manipulated within the genetic algorithm, directly influencing the optimization process and the accuracy of the final forecast. Given the need to optimize the transformation matrix L in the LMNN framework, the chromosome representation must capture the essential features of this matrix while allowing for efficient genetic operations such as crossover and mutation.

The chromosome in this context represents a candidate transformation matrix L, which plays a pivotal role in shaping the feature space and influencing the accuracy of the nearest neighbor calculations. For a dataset with d features, the transformation matrix L is a $d \times d$ matrix. The matrix is flattened into a single vector to represent this matrix within the genetic algorithm. This vector then serves as the chromosome, where each element corresponds to a specific entry in the matrix L.

For example, consider a scenario where the input data has three features $d = 3$. The transformation matrix L would be a 3×3 matrix with nine elements. This matrix is flattened into a vector of length nine in the chromosome representation. The vector is structured as $\{l_{11}, l_{12}, l_{13}, l_{21}, l_{22}, l_{23}, l_{31}, l_{32}, l_{33}\}$, where each l_{ij} represents a specific entry in the matrix. This encoding ensures that the entire structure of the transformation matrix is captured in the chromosome, allowing the genetic algorithm to explore the full space of possible transformations.

Within the genetic algorithm, the chromosome undergoes various operations that drive the optimization process. Crossover is one of the primary mechanisms for generating new candidate solutions by combining parts of two parent chromosomes. In the context of this flattened matrix representation, a single-point crossover might involve selecting a point along the vector and swapping the subsequent elements between two parent chromosomes. Alternatively, a two-point crossover might involve selecting two points and exchanging the genes between these points. These crossover operations allow the genetic algorithm to recombine different parts of the transformation matrices, potentially leading to better-performing solutions.

Mutation is another critical operation that introduces variability into the population, helping the algorithm avoid local minima. In the context of this method, a uniform mutation might involve randomly selecting a gene (an element of the matrix L and altering its value within a predefined range. This could mean adding or subtracting a small, fixed amount. A more sophisticated approach might involve Gaussian mutation, where a small, normally distributed random value is added to a selected gene. This type of mutation allows for subtle adjustments to the transformation matrix, which can fine-tune the model's performance.

The fitness of each chromosome, or candidate transformation matrix, is evaluated based on how well it minimizes the cost function defined in our method (equation (6)). The cost function reflects the performance of the transformation matrix in terms of how effectively it separates different classes in the feature space, thus directly impacting the load forecasting accuracy. Chromosomes that result in a lower cost function value are deemed more fit and are more likely to be selected for crossover and mutation in subsequent generations.

It might be necessary to impose constraints during the optimization process to ensure that the transformation matrix remains numerically stable and does not lead to overfitting. One common approach is regularizing the matrix L, perhaps by normalizing its Frobenius norm. This constraint would prevent the elements of L from becoming too large, which could otherwise lead to instability in the model. Regularization helps maintain a balance between model complexity and generalization, ensuring that the final solution performs well not only on the training data but also on unseen test data.

In summary, the chromosome representation in this Short-Term Load Forecasting method is a flattened vector of the transformation matrix L, capturing all the essential elements in a format suitable for genetic manipulation. The genetic algorithm operates on these chromosomes through crossover

and mutation, exploring the space of potential solutions and optimizing the transformation matrix to minimize the cost function. This process is guided by fitness evaluation, with constraints like normalization applied to ensure the stability and effectiveness of the resulting model.

E. Proposed LMNN Classifier

Assuming a set of points x_1, x_2, x_3,…,x_n which labels are y_i (i=1,2,…,n). The goal is to learn a linear transformation L leading to the following transformed distance:

$$D^2(\vec{x}_i,\vec{x}_j) = \left\|\mathrm{L}(\vec{x}_i-\vec{x}_j)\right\|^2 = (\vec{x}_i-\vec{x}_j)^T \mathrm{L}^T\mathrm{L}(\vec{x}_i-\vec{x}_j) \quad (3)$$

where L is a $d\times d$ matrix, and d is the dimension of the input vector. For each input x_i, k target neighbors are selected which are k inputs with the same label as x_i. The target neighbors can be identified as the k nearest neighbor with the same label as x_i. Fig. 2 illustrates the LMNN algorithm.

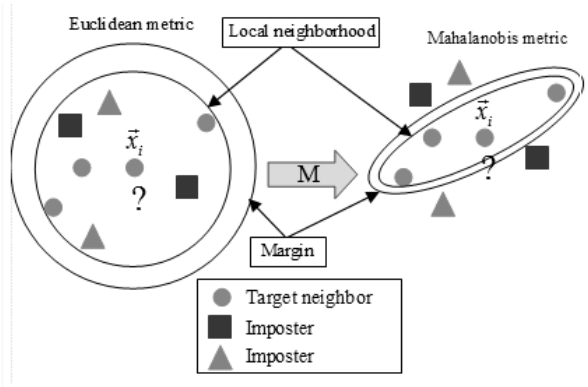


Fig. 2. Illustration of Large Margin Nearest Neighbor algorithm.

There are two terms in our objective function for the distance metric. The first term seeks to shorten the distance between any given data and its immediate neighbors. In contrast, the second term seeks to lengthen the distance between any given data and all other data not belonging to the same class.

These two terms compete because reducing the distance between samples reduces the first part while increasing it reduces the second. The large separation between each input and its target neighbors is penalized in the cost function's first section. The sum of squares of this distance is as follows when the input space is transformed linearly:

$$\varepsilon_1(\mathrm{L}) = \sum_{j\to i}\left\|\mathrm{L}|\vec{x}_i-\vec{x}_j|\right\|^2 \quad (4)$$

where L is a $d\times d$ matrix d is the dimension of the input vector, and $\vec{x}_i$ and $\vec{x}_j$ are inputs.

In the linear transformation of the input space, this expression generates a pulling force that pulls the target's surrounding neighbors toward it. The expression mentioned above does not penalize the large distance between all of the data bearing the same label, only the large distance between the inputs and their target neighbors. The second case is being penalized on purpose. Thus, the way that our method differs from many other distance metric approaches is that it penalizes great distances between neighbors.

The second part of the cost function penalizes the short distance between data with different labels:

$$\varepsilon_2(\mathrm{L}) = \sum_{i,j\to i}\sum_{l}(1-y_{il})\Big[1 + \left\|\mathrm{L}(\vec{x}_i-\vec{x}_j)\right\|^2 - \left\|\mathrm{L}(\vec{x}_i-\vec{x}_l)\right\|^2\Big]_+ \quad (5)$$

where $[z]_+ = \max(0,z)$ is the hinge loss. L is a $d\times d$ matrix, and $\vec{x}_i$ and $\vec{x}_j$ are inputs. If y_i=y_l then y_{il}=1, otherwise y_{il}=0;

The goal is to optimize the cost function to fine better neighbors and determine the data class. Our cost function becomes:

$$cost(\mathrm{L},X) = (1-\mu)\varepsilon_1(\mathrm{L}) + \mu\varepsilon_2(\mathrm{L}) = (1-\mu)\sum_{j\to i}\left\|\mathrm{L}|\vec{x}_i-\vec{x}_j|\right\|^2 + \mu\sum_{i,j\to i}\sum_{l}(1-y_{il})\Big[1 + \left\|\mathrm{L}(\vec{x}_i-\vec{x}_j)\right\|^2 - \left\|\mathrm{L}(\vec{x}_i-\vec{x}_l)\right\|^2\Big]_+ \quad (6)$$

where the positive constant μ changes the importance of those two terms.

LMNN uses semi-definite programming (SDP) to transform the distance metric learning problem into a convex problem [31]. The SDP is:

$$\text{Minimize}\sum_{ij}\eta_{ij}(\vec{x}_i-\vec{x}_j)^T\mathrm{M}(\vec{x}_i-\vec{x}_j) + c\sum_{ij}\eta_{ij}(1-\vec{y}_{il})\varepsilon_{ijl} \quad (7)$$

where M is the semi-definite matrix of the Mahalanobis metric, c is the control variable and $\varepsilon_{ijl}$is the slack variable for hinge loss.

With conditions:

$(\vec{x}_i-\vec{x}_l)^T\mathrm{M}(\vec{x}_i-\vec{x}_l) - (\vec{x}_i-\vec{x}_j)^T\mathrm{M}(\vec{x}_i-\vec{x}_j) \geq 1-\varepsilon_{ijl}$

$\varepsilon_{ijl} \geq 0$

$\mathrm{M} \succeq 0$

Equation (6)'s cost function expressed in terms of L is not convex. Elements of L employ the gradient descent approach to minimize this function. However, this strategy is prone to becoming stuck in local minima. The outcomes of this gradient descent method typically rely on initial L estimates. As a result, they might not be applied to many problems or applications. In order to overcome this issue and optimize the objective function for more accurate classification, first, we use a genetic algorithm to find the global optimal range of the objective function, and then by using the gradient descent method, it is precisely determined by obtaining the optimal point of the parameter L in the cost function.

The suggestion to increase the efficiency of the global optimization methods is to combine the local optimization methods with the global one, which has the advantage of increasing the speed along with dodging local optima traps. After L is obtained, the distance between the test data and the neighboring points is determined to determine the similarity, and it is calculated based on the type of neighbors of the data set.

Our method can be summarized as follows: first, we choose the appropriate k and divide the dataset into training and test data. The cost function is defined using equation (6). Then, using the genetic algorithm, L_g is optimized. The first

step of the genetic algorithm is to find an initial population and the rate of mutation and crossover. Until the end criterion is met, parents are chosen, crossover and mutation are carried out, a new generation of offspring is created, and their fitness value is calculated. These steps are repeated until the semi-optimum L_g is found. The optimum L is obtained by gradient descent. Initially, learning rate α is defined, and L_0=L_g is set. Steps of gradient descent are repeated until optimum L is found. Then, the distance between the test and all training data is calculated using equation (3).

F. Short-Term Load Forecasting Based on LMNN

The specified implementation process includes the following steps to create the short-term load forecasting model based on the suggested LMNN algorithm: (1) Choosing the value of *k*. The majority of the *k* nearest objects for a sample (*S*) in its associated feature space belonged to a particular category, and so did the sample. Then, based on the traits of the objects in this category, the suitable nearest neighbor parameter, *k*, is chosen. Because of these traits, patterns of similar electricity consumption will undoubtedly clump together in a particular area; (2) Building the output set and sample set. Calculate the distance between the predicted data point and the known data point based on the random distribution (to ensure that all electricity consumptions are considered, not just the local optima). The weight for each predicted data point is then equal to 1/distance. Ultimately, it would be possible to obtain the predicted value for each data point. (3) Analysis of forecasting accuracy. The root mean square error (RMSE) and the normalized mean square error (NMSE) are used to assess the forecasting accuracy [32]. Equations (8) and (9) are used to calculate them, respectively. The proposed model's reliability and accuracy would then be further verified using the forecasting results computed by the MATLAB simulation and the actual data values.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{8}$$

$$\mathrm{NMSE} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{9}$$

where y_i stands for the actual load value and $\hat{y}_i$ stands for the *i*th predicted load value. *n* is the total number of predicted loads, and $\bar{y}$ is the mean value of *n* actual load values. The data are divided into two samples: a quarterly sample and a monthly sample. In the quarterly sample, the first two months' data serve as samples that predict the third month's load values. In the monthly sample divided up by month, the data from the first three weeks are used to forecast the last week's load.

The flowchart of the algorithm is illustrated in Fig. 3, and the pseudocode of our method is as follows:

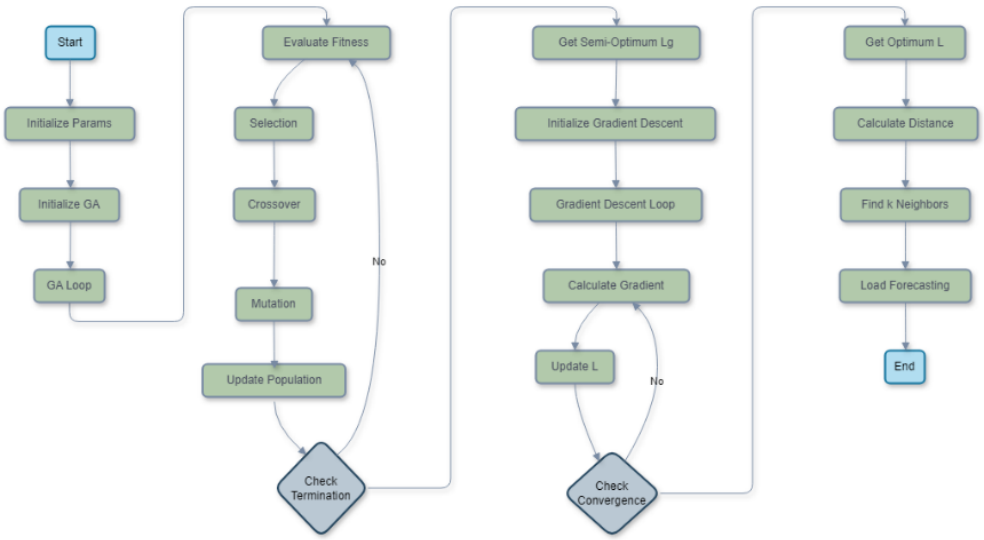


Fig. 3. Flowchart of our proposed method

```
# Pseudocode for Short-Term Load Forecasting using LMNN and Genetic Algorithm

# Step 1: Initialize parameters
Initialize k  # Number of nearest neighbors
Divide the dataset into training_data and test_data
Define cost function based on equation (6)

# Step 2: Genetic Algorithm to optimize Lg
# Initialize Genetic Algorithm parameters
Initialize population_size  # Size of the population
Initialize mutation_rate  # Rate of mutation
Initialize crossover_rate  # Rate of crossover
Initialize max_generations  # Maximum number of generations
Initialize Lg_population with random values
```

```
  # Step 3: Main Genetic Algorithm loop
  for generation in range(max_generations):
      # Evaluate fitness of each individual in the population
      for each individual in Lg_population:
          Calculate fitness value of individual using cost function
      # Selection: Choose parents for crossover
      parents = Select parents from Lg_population based on fitness values
      # Crossover: Generate new offspring
      offspring_population = []
      while size of offspring_population < population_size:
          parent1, parent2 = Randomly select two parents from parents
          child1, child2 = Perform crossover on parent1 and parent2 with
probability crossover_rate
          offspring_population.append(child1)
          offspring_population.append(child2)
      # Mutation: Apply mutation to offspring
      for each offspring in offspring_population:
          Apply mutation to offspring with probability mutation_rate
      # Replace the old population with the new population
      Lg_population = offspring_population
      # Check termination condition (e.g., max generations or convergence)
      if termination_condition_is_met:
          break

  # Step 4: Obtain semi-optimum Lg
  Lg = Best individual in the final Lg_population

  # Step 5: Gradient Descent to find the optimum L
  Initialize learning_rate α
  Set L0 = Lg
  Initialize tolerance and max_iterations for gradient descent

  # Step 6: Main Gradient Descent loop
  for iteration in range(max_iterations):
      # Calculate the gradient of the cost function with respect to L
      gradient = Calculate gradient of cost function at L0
      # Update L
      L_new = L0 - α * gradient
      # Check for convergence
      if |L_new - L0| < tolerance:
          break
      # Update L0 for the next iteration
      L0 = L_new
  # The optimal transformation matrix L is found
  L_optimal = L_new

  # Step 7: Calculate the distance between test data and all training data using
equation (3)
  for each test_point in test_data:
      distances = []
      for each training_point in training_data:
          distance = Calculate distance between test_point and training_point
using equation (3) and L_optimal
          distances.append(distance)
      # Sort distances to find the nearest neighbors
      nearest_neighbors = Sort distances and select the k nearest neighbors

  # Step 8: Use the nearest neighbors for Short-Term Load Forecasting
  # Forecast the load using the nearest neighbors' information
  forecasted_load = Predict load based on nearest_neighbors
```

Here's a detailed breakdown of the proposed algorithm:

Step 1: Data Preparation

Dataset Division: The dataset is divided into training and test sets. The training set is used to train the model, while the test set is used to evaluate its performance.

Step 2: Selection of *k*

Choosing *k*: The value of *k*, which represents the number of nearest neighbors to consider, is selected. This is a crucial parameter that affects the performance of the *k*-NN algorithm.

Step 3: Cost Function Definition

Cost Function: A cost function evaluates the model's performance. This function typically measures the error between the predicted and actual values.

Step 4: Genetic Algorithm Optimization

Initial Population: The genetic algorithm starts with a randomly generated population of potential solutions (chromosomes), each representing a set of parameters for the distance metric.

Fitness Evaluation: Each chromosome is evaluated based on its fitness, which is determined by how well it minimizes the cost function.

Selection, Crossover, and Mutation: The fittest chromosomes are selected to create a new generation. Crossover and mutation operators are applied to introduce variability and explore new solutions.

Iteration: This process is repeated until a stopping criterion is met, such as a maximum number of generations or convergence of the population.

Step 5: Gradient Descent Optimization

Initial Parameters: The best parameters identified by the genetic algorithm are used as the starting point for gradient descent.

Learning Rate: A learning rate (α) is defined to control the step size during optimization.

Parameter Update: The parameters are updated iteratively using the gradient of the cost function until the optimal parameters are found.

Step 6: Distance Calculation

Mahalanobis Distance: The optimized parameters are used to calculate the Mahalanobis distance between the test data and the training data. This distance metric accounts for the correlations between different features, making it more effective than traditional metrics like Euclidean distance.

Step 7: Classification and Forecasting

Classification: The algorithm is applied using the optimized distance metric. The algorithm identifies the *k* nearest neighbors of the test instance and assigns a category based on the majority class among these neighbors.

Forecasting: The final output is the forecasted load value based on the classification results.

IV. RESULTS and DISCUSSION

Simulations are implemented using MATLAB. The learning rate is set to 0.1, and μ is set to 0.7. For the genetic algorithm, the population size is 30, and the number of generations is 250 through trial and error to balance accuracy and speed. Crossover and mutation rates are set to 0.8 and 0.05, respectively. The selection type is a roulette wheel, the crossover type is a random pair and random point, and the mutation type is a random gene at a random chromosome. The gradient descent stop threshold is set to 0.0001. The

hourly electricity load data were obtained from the National Electricity Market of Australia for the entire 2021 calendar year. Load data encapsulates recurring patterns and is independent of weather conditions, which might not always be available or reliable.

A. Analysis for Different Values of k

Parameter *k* is a parameter for the LMNN algorithm that is used to classify samples based on the category label that occurs the most frequently among the *k* training samples closest to the chosen data point. The classification accuracy will decrease if the value of *k* is either too high or too low. When the value of *k* is low, the model is more complex, making it more likely to suffer from over-fitting, and the errors rise as a result. On the other hand, if *k* has a large value, the estimation errors would be reduced, but the errors would also increase, and the training data points' distance from the input data point would also impact the forecasting outcomes. As a result, the value of *k* is frequently set to a low value in general applications of the LMNN algorithm.

This study used different values to evaluate the experimental findings and choose an appropriate value for *k*. For instance, Tables I and II show, respectively, the determined suitable values of *k* for monthly samples and quarterly samples. Values above 3 have been shown to degrade the results overall.

TABLE I
Error Comparison for Various Values of *k* in Monthly Samples (Measured in Megawatts).

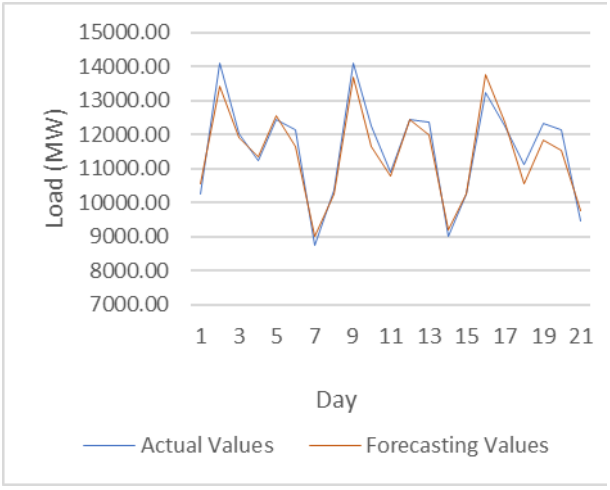
Month	*k=1*		*k=2*		*k=3*	
	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE
1	981.26	0.45	**673.93**	**0.21**	918.19	0.47
2	422.13	0.09	**394.30**	**0.08**	475.55	0.14
3	1243.86	1.09	**732.52**	**0.38**	1038.65	0.87
4	**477.60**	**0.23**	500.94	0.25	536.77	0.29
5	**399.31**	**0.14**	477.57	0.20	487.05	0.20
6	347.79	0.06	**264.86**	**0.03**	375.29	0.07
7	**657.47**	**0.37**	673.81	0.39	726.12	0.45
8	1318.91	1.43	949.29	0.74	**897.15**	**0.66**
9	**538.11**	**0.32**	558.66	0.35	637.55	0.59
10	2204.65	1.04	2167.92	1.00	**2166.19**	**0.99**
11	395.15	0.12	368.92	0.09	**264.19**	**0.05**
12	1523.11	1.02	**1272.35**	**0.71**	1277.71	0.72

TABLE II
Error Comparison for Various Values of *k* in Quarterly Samples (Measured in Megawatts).

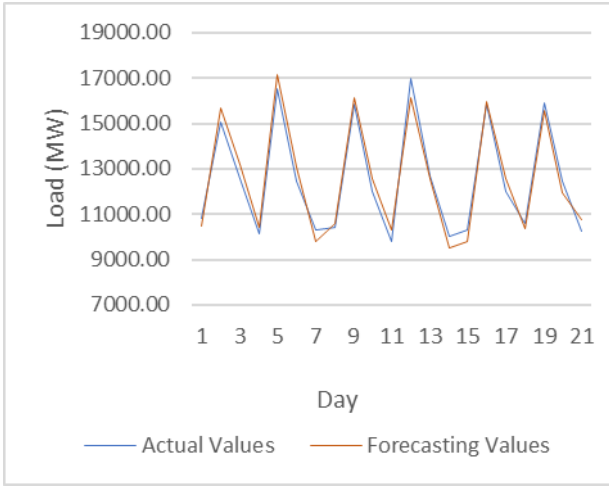
Season	*k=1*		*k=2*		*k=3*	
	RMSE	NMSE	RMSE	NMSE	RMSE	NMSE
1	998.92	0.55	**986.24**	**0.53**	993.68	0.54
2	1648.50	0.64	**1574.99**	**0.52**	1677.41	0.59
3	**572.34**	**0.17**	636.54	0.21	755.07	0.29
4	1320.92	1.14	936.19	0.58	**855.62**	**0.48**

It shows that when *k* was set to 2, the error is reasonably small for both cases.

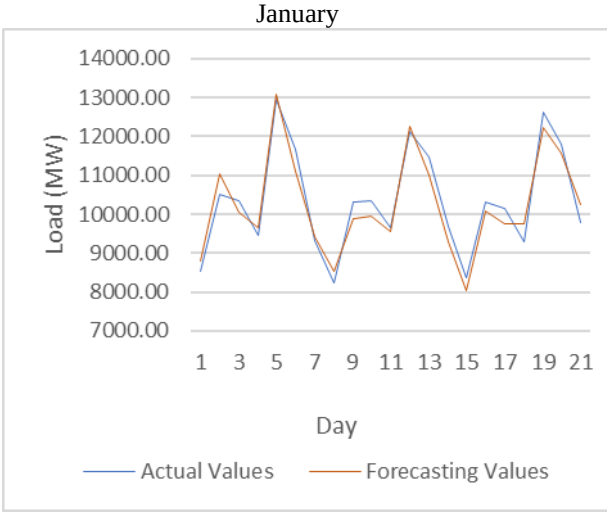
The proposed LMNN model performed the forecasting processes and the associated results. The load data were obtained from the National Electricity Market of Australia for the year 2021. The data for this paper were gathered by dividing each day into three equal parts and averaging each. The electricity forecasting value for one week (21 eight-hour splits) obtained from the LMNN model is shown in Fig. 4.



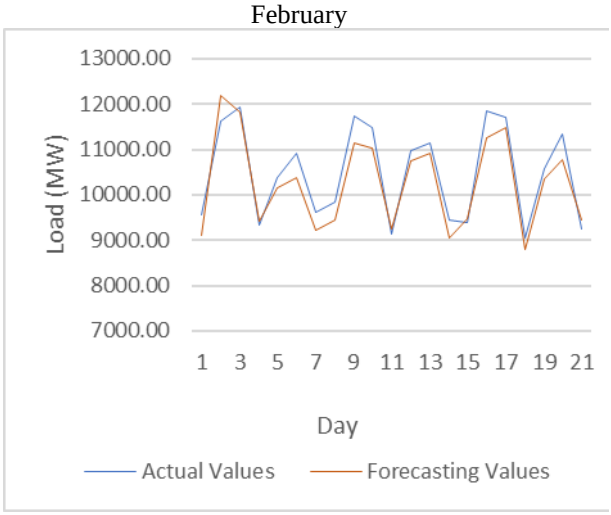
January



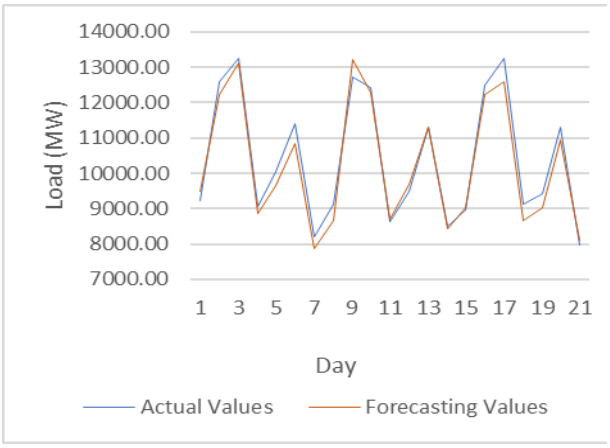
February



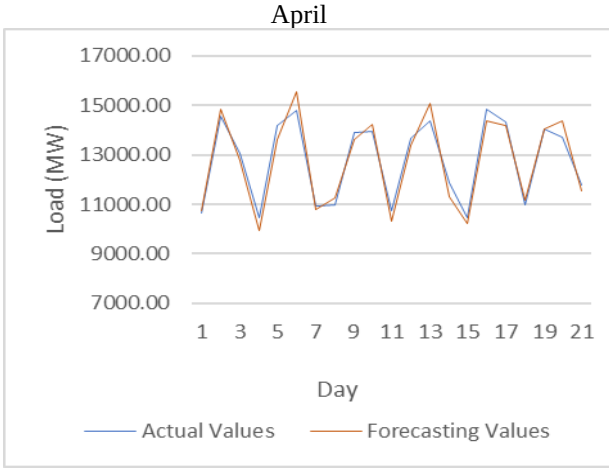
March



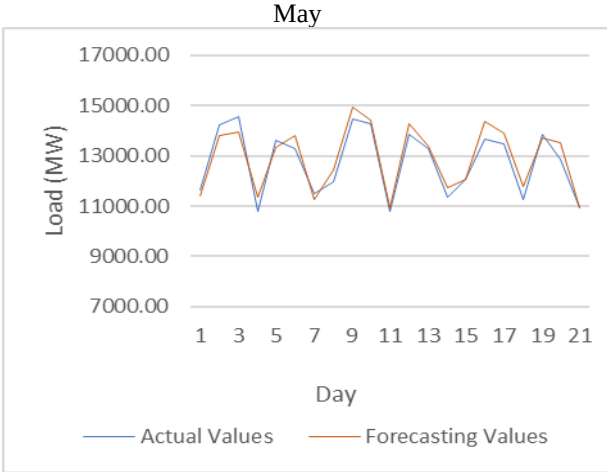
April



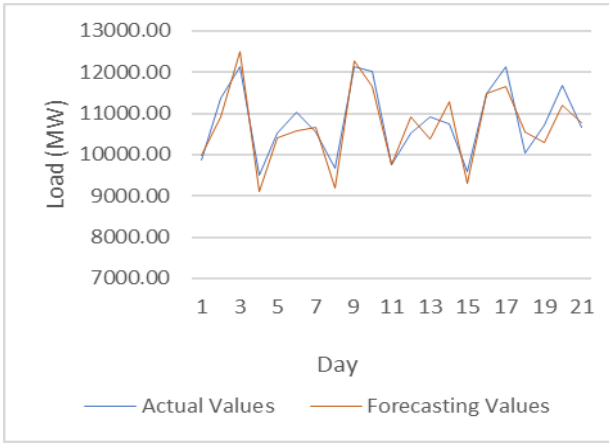
May



June



July



August

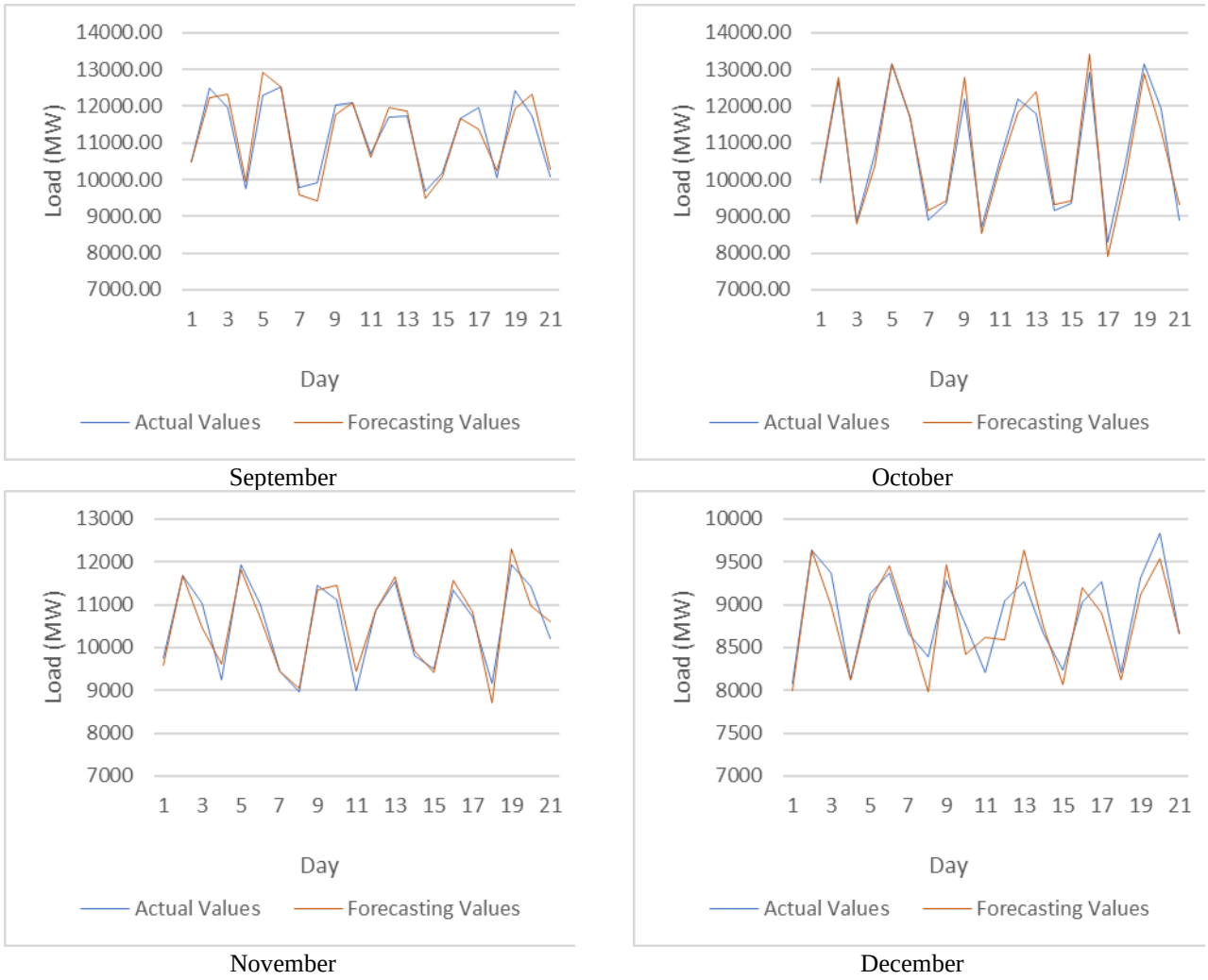


Fig. 4. Forecasting results for the last week of each month.

Fig. 4 demonstrates that overall, there is a reliable trend between the actual data and the forecasted data. It shows that the proposed LMNN model is appropriate for short-term prediction despite some errors. Each day starts with the lowest load at the first third of the day, which increases in the middle of the day, and then comes back down at the last third of the day while staying above the first third, which creates repeated patterns in these fig.s.

B. Forecasting Results Comparison

The Autoregressive-Moving Average model [33] and the Back-Propagation Neural Network model [34] were chosen for comparison analysis to show the superiority of the proposed model. Table III displays the results of the comparison between models using root mean square error (RMSE), normalized mean square error (NMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE) [35].

Time series data analysis and forecasting are done statistically using the Autoregressive Moving Average (ARMA) model. It consists of moving average (MA) and autoregression (AR). An observation in a time series and a specific number of lagged observations (previous values) are related to an autoregressive component. It suggests that the series' current value is a linear combination of its earlier values. The moving average component represents the relationship between the current observation and the residual errors from a moving average model applied to lag observations. An ARMA model seeks to represent the temporal dependencies in a time series dataset by merging these two elements. Various time series data, including stock prices, temperature variations, economic indicators, and more, can be modeled and forecasted using ARMA models.

Backpropagation Neural Network (BPNN) is an artificial neural network designed to learn and recognize patterns in data. It comprises interconnected nodes arranged in layers, allowing data to move from input to output nodes via hidden layers. Using a training algorithm known as backpropagation, BPNNs gradually minimize errors over several iterations by adjusting the network's weights and biases in response to variations between predicted and actual outputs. Because of their feedforward network architecture, BPNNs can perform well in various tasks like pattern recognition, regression, and classification. Because of their capacity to represent intricate relationships within data, they find applications in a wide range of industries, including finance, healthcare, image and speech recognition, and more. The number of layers was chosen as 3 with 10 neurons, as recommended by the original article. The Tansig and Logsig functions were also chosen for the hidden and output layers, respectively, for the same reason.

TABLE III
Comparison of Four Forecasting Models (LMNN, k-NN, ARMA, and BPNN). Unit: MW.

Month	*LMNN*				*k-NN*			
	RMSE	NMSE	MAE	MAPE (%)	RMSE	NMSE	MAE	MAPE (%)
1	649.91	0.38	1.15	4.29	946.28	0.43	1.30	6.58
2	379.60	0.71	1.35	3.88	407.08	0.09	1.84	6.33
3	706.40	0.33	1.09	3.97	1199.52	1.05	1.21	5.97
4	483.08	0.96	1.08	4.39	460.57	0.22	1.20	8.81
5	460.55	0.09	1.05	4.18	385.08	0.13	1.14	6.32
6	300.88	0.69	1.24	4.57	335.39	0.06	1.39	6.85
7	649.79	0.38	1.02	4.36	634.03	0.35	1.14	6.64
8	915.45	0.71	0.98	4.15	1271.89	1.38	1.03	7.33
9	538.74	0.33	0.94	3.94	518.92	0.31	1.05	5.91
10	2090.63	0.96	1.47	4.87	2126.05	1.00	1.64	7.72
11	355.77	0.09	1.22	4.45	381.06	0.11	1.36	9.17
12	1226.99	0.69	1.12	4.08	1468.80	0.99	1.15	6.12
Month	ARMA				BPNN			
	RMSE	NMSE	MAE	MAPE (%)	RMSE	NMSE	MAE	MAPE (%)
1	1116.87	0.42	1.45	16.87	2509.89	0.45	1.91	22.50
2	1083.87	0.61	1.36	15.47	2343.75	0.64	1.90	21.38
3	867.28	0.39	1.22	14.64	1899.53	0.63	1.49	18.72
4	808.59	0.38	1.13	14.20	2694.52	0.51	2.05	25.61
5	818.04	0.43	1.18	13.42	1397.62	1.05	1.23	14.28
6	1138.03	0.53	1.63	16.48	4405.28	0.54	3.36	33.77
7	835.01	0.52	1.19	12.34	1498.19	0.45	1.12	12.21
8	924.46	0.45	1.01	11.77	1565.69	0.64	1.30	14.92
9	725.88	0.40	1.01	12.44	1547.61	0.63	1.28	15.40
10	1730.04	0.39	1.85	99.91	3296.65	0.51	2.26	87.41
11	958.97	0.32	1.25	14.27	2612.32	1.05	1.76	19.50
12	1375.49	0.37	1.20	15.54	1651.78	0.54	1.25	17.91

V.CONCLUSION

This study built a new short-term load forecasting model using the large margin nearest neighbor algorithm. This proposed model was then used to perform the actual short-term load forecasting task. It would be very beneficial to suggest new load prediction techniques and enhance existing ones. Conventional techniques frequently rely significantly on the approach taken to determine how similar two samples are. In fixing the LMNN's premature convergence issue and refining the cost function to determine the separations between data, we also introduced a cost function to compute data similarities. The genetic algorithm was utilized to narrow the solution space's range, and gradient descent was then employed to determine the cost function's ideal parameter. The following are a couple of findings: (1) It is evident that the proposed LMNN model has higher forecasting accuracy demonstrated via forecasting error comparison. (2) The proposed model's ability to predict is superior to that of the ARMA model and the BPNN model when compared. It can better meet the development needs of today's grids and control systems.

Innovations in our proposed approach include:

- *Hybrid Optimization Approach:* The integration of genetic algorithms with gradient descent in the LMNN model addresses the limitations of traditional algorithms, improving adaptability and performance.
- *Distance Learning Enhancement*: The optimization of the Mahalanobis distance metric within LMNN allows for better classification and forecasting of electricity loads, accommodating dynamic consumption patterns.
- *Real-Time Data Integration*: Utilization of data analytics for processing load data enhances forecasting accuracy, showcasing the feasibility of real-time applications in modern power systems.
- *Local Optima Avoidance*: The combination of GA and gradient descent helps avoid local optima traps, ensuring a more reliable search for the best parameters.
- *Adaptability*: The method is designed to adapt to dynamic energy consumption patterns, making it suitable for various modern power systems.

REFERENCES

[1] Ahmed and M. Khalid, “A review on the selected applications of forecasting models in renewable power systems,” *Renewable and Sustainable Energy Reviews*, vol. 100, pp. 9–21, Oct. 2018.

[2] O. Rubasinghe *et al*., “Highly accurate peak and valley prediction short-term net load forecasting approach based on decomposition for power systems with high PV penetration,” *Applied Energy*, vol. 333, p. 120641, Jan. 2023.

[3] S. H. Rafi, N. Nahid-Al-Masood, S. R. Deeba, and E. Hossain, “A Short-Term load forecasting method using integrated CNN and LSTM network,” *IEEE Access*, vol. 9, pp. 32436–32448, Jan. 2021.

[4] H. J. Sadaei, P. C. De Lima E Silva, F. G. Guimarães, and M. H. Lee, “Short-term load forecasting by using a combined method of convolutional neural networks and fuzzy time series,” *Energy*, vol. 175, pp. 365–377, Mar. 2019.

[5] P. Ray, S. K. Panda, and D. P. Mishra, “Short-Term load forecasting using genetic algorithm,” in *Advances in intelligent systems and computing*, 2018, pp. 863–872.

[6] P. Cunningham and S. J. Delany, “K-Nearest Neighbour Classifiers - a

tutorial," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–25, Jul. 2021.
[7] F. Martínez, M. P. Frías, M. D. Pérez, and A. J. Rivera, "A methodology for applying k-nearest neighbor to time series forecasting," *Artificial Intelligence Review*, vol. 52, no. 3, pp. 2019–2037, Nov. 2017.
[8] Y. Dong, X. Ma, and T. Fu, "Electrical load forecasting: A deep learning approach based on K-nearest neighbors," *Applied Soft Computing*, vol. 99, p. 106900, Nov. 2020.
[9] S. Curteanu, F. Leon, A.-M. Mircea-Vicoveanu, and D. Logofătu, "Regression Methods Based on Nearest Neighbors with Adaptive Distance Metrics Applied to a Polymerization Process," *Mathematics*, vol. 9, no. 5, p. 547, Mar. 2021.
[10] M. Zhang, H. Li, and X. Deng, "Inferential statistics and machine learning models for Short-Term Wind Power Forecasting," *Energy Engineering*, vol. 119, no. 1, pp. 237–252, Nov. 2021.
[11] T. Ashfaq and N. Javaid, "Short-Term Electricity Load and Price Forecasting using Enhanced KNN," *2019 International Conference on Frontiers of Information Technology (FIT)*, pp. 266–2665, Dec. 2019.
[12] M. Gómez-Omella, I. Esnaola-Gonzalez, S. Ferreiro, and B. Sierra, "k-Nearest patterns for electrical demand forecasting in residential and small commercial buildings," *Energy and Buildings*, vol. 253, p. 111396, Aug. 2021.
[13] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using Deep Neural Networks," *42nd Annual Conference of the IEEE Industrial Electronics Society*, pp. 7046–7051, Oct. 2016.
[14] S. Ryu, J. Noh, and H. Kim, "Deep neural network based demand side short term load forecasting," *Energies*, vol. 10, no. 1, p. 3, Dec. 2016.
[15] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-Term residential load forecasting based on LSTM Recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, Sep. 2017.
[16] A. S. Khwaja, A. Anpalagan, M. Naeem, and B. Venkatesh, "Joint bagged-boosted artificial neural networks: Using ensemble machine learning to improve short-term electricity load forecasting," *Electric Power Systems Research*, vol. 179, p. 106080, Nov. 2019.
[17] T. Bashir, C. Haoyong, M. F. Tahir, and Z. Liqiang, "Short term electricity load forecasting using hybrid prophet-LSTM model optimized by BPNN," *Energy Reports*, vol. 8, pp. 1678–1686, Jan. 2022.
[18] J. Li, S. Zhang, and Z. Yang, "A wind power forecasting method based on optimized decomposition prediction and error correction," *Electric Power Systems Research*, vol. 208, p. 107886, Feb. 2022.
[19] F. E. Bezerra, F. Grassi, C. G. Dias, and F. H. Pereira, "A PCA-based variable ranking and selection approach for electric energy load forecasting," *International Journal of Energy Sector Management*, vol. 16, no. 6, pp. 1172–1191, Feb. 2022.
[20] S. S. Subbiah and J. Chinnappan, "Deep learning based short term load forecasting with hybrid feature selection," *Electric Power Systems Research*, vol. 210, Sep 2022.
[21] N. Neeraj, J. Mathew, M. Agarwal, and R. K. Behera, "Long short-term memory-singular spectrum analysis-based model for electric load forecasting," *Electrical Engineering*, vol. 103, no. 2, pp. 1067-1082, Apr. 021.
[22] V. Jensen, F. M. Bianchi, and S. N. Anfinsen, "Ensemble conformalized quantile regression for probabilistic time series forecasting," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, Nov. 2022.
[23] J. Moon, S. Rho, and S. W. Baik, "Toward explainable electrical load forecasting of buildings: A comparative study of tree-based ensemble methods with Shapley values," *Sustainable Energy Technologies and Assessments*, vol. 54, p. 102888, Nov. 2022.
[24] W. Xing and Y. Bei, "Medical Health big data classification based on KNN Classification Algorithm," *IEEE Access*, vol. 8, pp. 28808–28819, Nov. 2019.
[25] L. Wang, X. Liu, J. Yi, Y. Jiang, and C.-J. Hsieh, "Provably robust metric learning," *Neural Information Processing Systems*, vol. 33, pp. 19302–19313, Jan. 2020.
[26] L. Yang and R. Jin, "Distance metric learning: A comprehensive survey," *Michigan State Universiy*, 2006.
[27] S. R. Silva, T. Vieira, D. Martínez, and A. Paiva, "On novelty detection for multi-class classification using non-linear metric learning," *Expert Systems With Applications*, vol. 167, p. 114193, Nov. 2020.
[28] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, Oct. 2020.
[29] S. H. Haji and A. M. Abdulazeez, "Comparison of optimization techniques based on gradient descent algorithm: A review*," PalArch's Journal of Archaeology of Egypt/Egyptology*, vol. 18, no. 4, pp. 2715-2743, 2021.
[30] K. Chandra, A. Xie, J. Ragan-Kelley, and E. Meijer, "Gradient Descent: The Ultimate Optimizer," *arXiv.org*, Sep. 2019.
[31] J. Zhang, Y. Chen, and Y. Zhai, "Zero-Shot classification based on word vector enhancement and distance metric learning," *IEEE Access*, vol. 8, pp. 102292–102302, Jan. 2020.
[32] T. O. Hodson, "Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not," *Geoscientific Model Development*, vol. 15, no. 14, pp. 5481–5487, Jul. 2022, doi: 10.5194/gmd-15-5481-2022.
[33] J. Lu, J. Peng, J. Chen, and K. A. Sugeng, "Prediction method of autoregressive moving average models for uncertain time series," *International Journal of General Systems*, vol. 49, no. 5, pp. 546–572, Apr. 2020.
[34] H. K. Yadav, Y. Pal, and M. M. Tripathi, "Short-term PV power forecasting using empirical mode decomposition in integration with back-propagation neural network," *Journal of Information and Optimization Sciences*, vol. 41, no. 1, pp. 25–37, Jan. 2020.
[35] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, p. e623, Jul. 2021.