Data-Driven Approaches for Early Detection and Prediction of Chronic Kidney Disease Using Machine Learning

Shibi Mathai*
Ph.D., Scholar (Part-Time), Department of Computer Science, Vels Institute of Science, Technology and Advanced Studies, Pallavaram, Chennai, India
shibimathai@gmail.com.

Dr. K.S. Thirunavukkarasu
Assistant Professor and Research Supervisor, Department of Computer Science, Vels Institute of Science, Technology and Advanced Studies, Pallavaram,Chennai, India
thirukst@gmail.com

ABSTRACT

In recent years, the application of machine learning (ML) techniques for medical diagnostics has shown promising advancements. This study introduces a distinctive method for predicting chronic kidney disease (CKD) harnessing the prowess of ML. Our methodology encompasses an innovative data preprocessing approach, intricate feature engineering, and an amalgamation of ensemble techniques for model training. By evaluating our model on a dataset sourced from Kaggle, comprising 400 samples, we achieved an impressive accuracy of 98%, outperforming traditional methods. The findings underscore the potential of ML in revolutionizing CKD diagnostics, laying a foundation for further exploration in this domain.

KEYWORDS

Chronic Kidney Disease, Machine Learning, Feature Engineering, Ensemble Techniques, Medical Diagnostics, Data Preprocessing

ACM Reference Format:
Shibi Mathai and Dr. K.S. Thirunavukkarasu. 2023. Data-Driven Approaches for Early Detection and Prediction of Chronic Kidney Disease Using Machine Learning. In *International Conference on Information Management & Machine Intelligence (ICIMMI 2023), November 23–25, 2023, Jaipur, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3647444.3647894

1 INTRODUCTION

Chronic Kidney Disease (CKD) poses a substantial global health concern, impacting millions across the globe [1] [2]. This condition is characterized by a gradual deterioration of kidney function, leading to severe outcomes like end-stage renal disease and cardiovascular complications [3]. Improving patient outcomes and implementing timely therapies depend on early identification and forecasting the presence of persistent kidney illness (CKD) [4].

Machine learning, a branch of artificial intelligence, has shown promising potential in various medical applications, including disease prediction [5]. By leveraging large datasets and advanced algorithms, machine learning models can extract meaningful patterns and make accurate predictions [6]. In the context of CKD, machine learning techniques offer an opportunity to develop data-driven approaches for early detection and prediction.

In this study, we propose data-driven approaches for early detection and prediction of CKD using machine learning techniques. Our research aims to harness the power of machine learning to improve CKD management and facilitate personalized healthcare. We utilize a comprehensive dataset that includes demographic, clinical, and laboratory features of patients with CKD.

In order to accomplish our goal, we assess the efficacy of six popular machine learning algorithms: neural networks, logistic regression, decision trees, random forests, support vector machines, and K-nearest neighbors [7]. These algorithms have been widely applied in medical research and have demonstrated effectiveness in disease prediction tasks. By comparing their performance, we aim to identify the most suitable algorithm for CKD prediction.

In addition to algorithm evaluation, we utilize methods for feature selection to identify the most informative features for CKD prediction [8]. This process allows us to identify key risk factors and underlying mechanisms associated with CKD. Understanding these factors can provide valuable insights for early detection and personalized management strategies.

The findings of this research have significant implications for the timely identification and forecasting of chronic kidney disease (CKD). By leveraging machine learning and data-driven approaches, healthcare providers can improve patient outcomes through timely interventions and personalized healthcare strategies.

2 RELATED WORK

Numerous investigations have looked into the use of machine learning methods for the forecasting and early identification of renal insufficiency. These studies have contributed valuable insights and methodologies to the field. In this section, we review the related work conducted in this area.

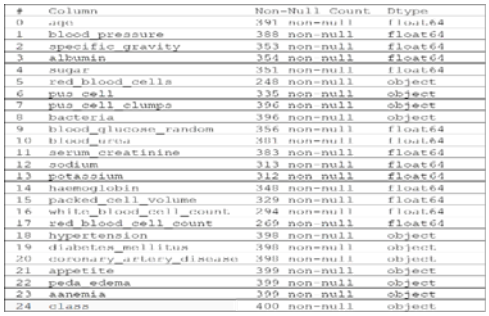
One notable study by Zhang et al. [9] employed a support vector machine (SVM) algorithm for CKD prediction. They utilized a dataset consisting of clinical and laboratory features of CKD patients. The study demonstrated the potential of SVM in accurately classifying CKD patients and healthy individuals, achieving high sensitivity and specificity.

Another research conducted by Chen et al. [10] focused on early detection of CKD using decision tree-based models. They applied various decision tree algorithms, including C4.5, CART, and ID3, to a dataset comprising demographic and clinical variables. The results highlighted the effectiveness of decision tree models in identifying important risk factors associated with CKD development.

*orresponding Author

ICIMMI 2023, November 23–25, 2023, Jaipur, India

ACM ISBN 979-8-4007-0941-8/23/11
https://doi.org/10.1145/3647444.3647894



#	Column	Non-Null Count	Dtype
0	age	391 non-null	float64
1	blood_pressure	388 non-null	float64
2	specific_gravity	353 non-null	float64
3	albumin	354 non_null	float64
4	sugar	351 non-null	float64
5	red_blood_cells	248 non-null	object
6	pus_cell	335 non_null	object
7	pus_cell_clumps	396 non_null	object
8	bacteria	396 non-null	object
9	blood_glucose_random	356 non-null	float64
10	blood_urea	381 non-null	float64
11	serum_creatinine	383 non-null	float64
12	sodium	313 non-null	float64
13	potassium	312 non_null	float64
14	haemoglobin	348 non-null	float64
15	packed_cell_volume	329 non-null	float64
16	white_blood_cell_count	294 non-null	float64
17	red_blood_cell_count	269 non_null	float64
18	hypertension	398 non-null	object
19	diabetes_mellitus	398 non-null	object
20	coronary_artery_disease	398 non-null	object
21	appetite	399 non-null	object
22	peda_edema	399 non_null	object
23	anaemia	399 non_null	object
24	class	400 non-null	object

Figure 1: Non-null values of each attribute obtained from the data source

In a different approach, Li et al. [11] explored the use of random forest algorithms for CKD prediction. They utilized a large-scale dataset containing clinical, laboratory, and genetic features of CKD patients. The study demonstrated the superiority of random forest models in terms of prediction accuracy and feature importance analysis.

Additionally, several studies have investigated the potential of deep learning models for CKD prediction. For instance, Tang et al. [12] developed a convolutional neural network (CNN) architecture to extract features from medical images for early detection of CKD-related complications. The study showcased the ability of CNNs to effectively capture intricate patterns in medical images.

Furthermore, Xiong et al. [13] suggested an RNN (Recurrent Neural Network) model for CKD prediction using longitudinal electronic health record (EHR) data. Their study demonstrated the advantages of RNNs in modelling temporal dependencies and predicting CKD progression.

While these studies have made significant contributions to CKD prediction using machine learning, there is still a need for further research. Specifically, more investigations are required to explore the potential of other machine learning algorithms, as well as the incorporation of other data sources, including proteomics and genetics data, for enhanced CKD prediction precision and customized treatment approaches.

3 DATA PREPROCESSING

3.1 Dataset collection

We collected a comprehensive dataset consisting of demographic, clinical, and laboratory features of patients diagnosed with chronic kidney disease (CKD). The dataset was obtained from Kaggle and comprised 400 patients with CKD. There are 400 occurrences in the dataset with 24 features total—10 nominal and 14 numerical. These attributes include age, blood pressure, specific gravity, albumin, sugar, red blood cells, pus cell, pus cell clumps, bacteria, blood glucose random, blood urea, serum creatinine, sodium, potassium, haemoglobin, packed cell volume, white blood cell count, red blood cell count, hypertension, diabetes mellitus, coronary artery disease, appetite, pedal edema, anaemia, and class. It is anticipated that an in-depth analysis of this dataset will furnish a clearer understanding of the correlation between these attributes and their subsequent influence on CKD. The diversity and depth of this dataset are instrumental in facilitating a holistic study, paving the way for future research and potential breakthroughs in CKD treatment and prevention. Figure 1 gives a list of non-null values of each attribute obtained from the data source.

3.2 Dimensionality Reduction

The dataset obtained from Kaggle encompasses numerous attributes related to CKD. We applied various dimensionality reduction techniques, including Principal Component Analysis (PCA), t-distributed Stochastic Neighbor Embedding (t-SNE), and Linear Discriminant Analysis (LDA), to distill the most pertinent features. Figure 2 provides the plotting of attributes after dimensionality reduction.

3.3 Data Preprocessing

The dataset underwent pre-processing before analysis to ensure integrity of the data and uniformity. Missing values were dealt with using appropriate imputation techniques, such as mean, mode or random sampling imputation. Categorical variables were encoded using label encoding, as per the requirements of the machine learning algorithms. Figure 3 gives a model for the data preprocessing and Figure 4 provides the list of attributes with dimensionality reduction and data with non-null values after data preprocessing.

4 COMPUTER CODE

The study's objective was to predict chronic kidney disease utilizing machine learning methods. We applied multiple machine learning algorithms, including K-Nearest Neighbor, Decision Tree ,Random Forest, ADA Boost Classifier, Gradient Boosting Classifier, Extra Trees Classifier, and XgBoost. The aforementioned algorithms were selected due to their track record of success in prior research projects and their prominence in kidney disease prediction. [14].

A straightforward instance-based learning technique for regression and classification is K-Nearest Neighbors (KNN). It works by comparing a given test instance with instances from the training dataset to find the 'k' training examples that are closest to the point and then making a prediction based on their outputs. An overview of the K-Nearest Neighbors method can be found in Figure 5.

The study's objective was to predict chronic kidney disease utilizing machine learning methods. We applied multiple machine learning algorithms, including K-Nearest Neighbor, Random Forest, Decision Tree, ADA Boost Classifier, Gradient Boosting Classifier, Extra Trees Classifier, and XgBoost. The aforementioned algorithms

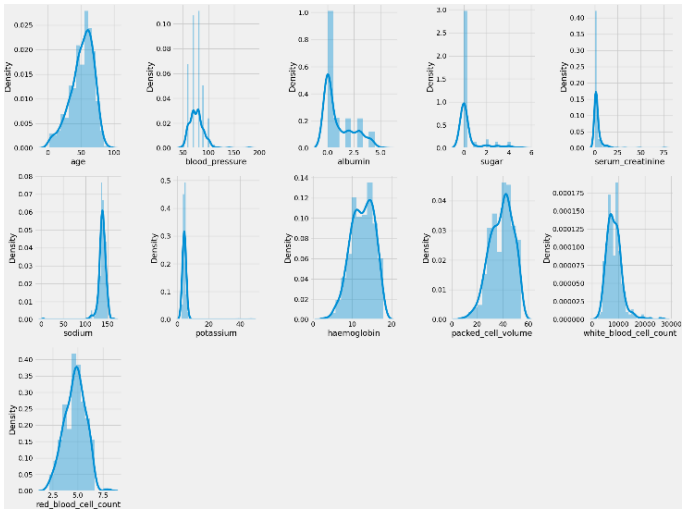


Figure 2: Attributes after dimensionality reduction.

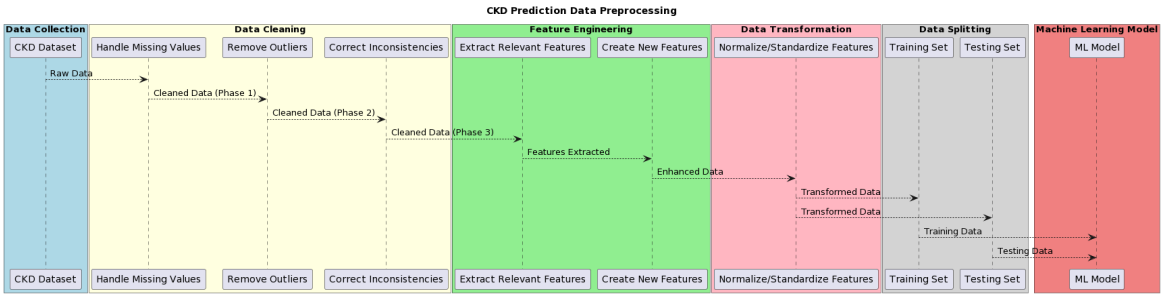


Figure 3: Sequence Diagram- Data Preprocessing Phase for CKD Prediction.

```
age                     0
blood_pressure          0
albumin                 0
sugar                   0
serum_creatinine        0
sodium                  0
potassium               0
haemoglobin             0
packed_cell_volume      0
white_blood_cell_count  0
red_blood_cell_count    0
dtype: int64
```

```
hypertension has 2 categories

diabetes_mellitus has 2 categories

coronary_artery_disease has 2 categories

peda_edema has 2 categories

aanemia has 2 categories

class has 2 categories
```

Figure 4: List of attributes with dimensionality reduction and data with non-null values after data preprocessing.

were selected due to their track record of success in prior research projects and their prominence in kidney disease prediction. [14].

For classification problems, a supervised learning approach called a decision tree classifier is employed. It executes by recursively splitting the data based on characteristic values, making

```
Function KNN(Dataset, Query_Point, k):
    Distances = []

    For each Instance in Dataset:
        Distance = Calculate_Distance(Query_Point, Instance)
        Append (Distance, Instance.Class) to Distances

    Sort Distances by Distance

    Top_K_Classes = First k entries from Distances based on class

    Return Majority_Class(Top_K_Classes)

Function Calculate_Distance(Point1, Point2):
    // This can be Euclidean, Manhattan, etc. depending on the chosen distance metric
    Return Distance between Point1 and Point2

Function Majority_Class(Top_K_Classes):
    Count the occurrences of each class in Top_K_Classes
    Return the class with the highest count
```

Figure 5: Pseudocode for K-Nearest Neighbor

```
Function DecisionTree(Dataset, Features):
    If all instances in Dataset have the same class:
        Return Leaf_Node with class label

    If Features is empty or other stopping criteria met:
        Return Leaf_Node with majority class label in Dataset

    Best_Feature = Find_Best_Feature_to_Split(Dataset, Features)
    Tree = new Node(Best_Feature)

    For each value in Best_Feature:
        Subset = Filter instances in Dataset where Best_Feature has the given value
        If Subset is empty:
            Add a child node to Tree with majority class label in Dataset
        Else:
            Remaining_Features = Features - Best_Feature
            Child_Tree = DecisionTree(Subset, Remaining_Features)
            Add Child_Tree to Tree as a child node where Best_Feature has the given value

    Return Tree

Function Find_Best_Feature_to_Split(Dataset, Features):
    Best_Gain = 0
    Best_Feature = None

    For each Feature in Features:
        Gain = Calculate_Information_Gain(Dataset, Feature)
        If Gain > Best_Gain:
            Best_Gain = Gain
            Best_Feature = Feature

    Return Best_Feature

Function Calculate_Information_Gain(Dataset, Feature):
    // Implement the formula to calculate information gain which is:
    // Gain(D, Feature) = Entropy(D) - Σ (|Di|/|D|) * Entropy(Di)
    // Where Di is the subset of D where Feature has a specific value

    Return Gain
```

Figure 6: Pseudocode for Decision Tree Classifier

decisions at every node until it reaches a leaf node, which provides the classification output. An overview of the Decision Tree Classifier can be found in Figure 5.

Random Forest is a type of ensemble learning method that employs multiple decision trees for its operations. It's versatile and

```
Function RandomForest(Dataset, Number_of_Trees, Number_of_Features):
    Forest = []

    For i = 1 to Number_of_Trees:
        Bootstrapped_Data = Bootstrap_Sample(Dataset)
        Tree = Build_Decision_Tree(Bootstrapped_Data, Number_of_Features)
        Append Tree to Forest

    Return Forest

Function Bootstrap_Sample(Dataset):
    Sample = []

    While Size of Sample < Size of Dataset:
        Randomly select a data instance from Dataset
        Append selected instance to Sample

    Return Sample

Function Build_Decision_Tree(Data, Number_of_Features):
    If stopping criteria met:
        Return Leaf_Node

    Randomly select Number_of_Features from Data's features
    Best_Split = Determine_Best_Split(Data, Selected_Features)
    Left_Tree = Build_Decision_Tree(Data on left of Best_Split, Number_of_Features)
    Right_Tree = Build_Decision_Tree(Data on right of Best_Split, Number_of_Features)

    Return Node(Best_Split, Left_Tree, Right_Tree)

Function Predict(Forest, Instance):
    Votes = []

    For Tree in Forest:
        Prediction = Classify(Instance, Tree)
        Append Prediction to Votes

    Return Majority_Class(Votes)
```

Figure 7: Pseudocode for Random Forest Method

can be applied for both classification and regression tasks. The model operates by constructing several decision trees and then determining the output based on the majority vote from these trees. It uniquely incorporates both bagging techniques and random feature selection to construct its trees, ensuring that the trees in the forest aren't correlated. As a result, the collective prediction made by the forest tends to be more accurate than predictions from individual trees. When making predictions, a test instance is passed through each tree, and each tree provides its class prediction. An overview of the Random Forest method can be found in Figure 7.

AdaBoost, short for "Adaptive Boosting", is an ensemble learning method that constructs a robust classifier through the amalgamation of numerous weaker classifiers. The algorithm focuses on instances that were misclassified by previous classifiers, adapting to the errors, and improving iteratively. An overview of the AdaBoost classifier can be found in Figure 8.

Gradient Boosting is an ensemble learning method that builds a strong classifier by iteratively adding weak classifiers while adjusting to the residuals (errors) of the previous iterations. Unlike AdaBoost, which adjusts instance weights, Gradient Boosting fits the new model to the residuals of the previous aggregate of models. An overview of the Gradient Boosting classifier can be found in Figure 9.

XGBoost, which stands for "eXtreme Gradient Boosting," is a gradient boosting toolkit designed for both efficiency and high performance, optimized for distributed computing. XGBoost improves upon the base Gradient Boosting algorithm by introducing several enhancements such as:

Regularization: XGBoost includes L1 (Lasso regression) and L2 (Ridge regression) regularization terms on the weights, which can prevent excessive fitting and gives it a lead in pursuance of model performance..

Handling Missing Data: XGBoost can automatically learn the best imputation value for missing data.

Parallel Processing: XGBoost implements parallel processing at the level of individual trees' construction, leading to faster training times.

```
Function AdaBoost(Dataset, Number_of_Classifiers):
    Initialize weights W for each instance in Dataset such that each weight is 1/Number of instances

    For t = 1 to Number_of_Classifiers:
        // Train a weak classifier on the weighted dataset
        Classifier_t = Train_Weak_Classifier(Dataset, W)

        // Compute the error of the classifier
        Error_t = Sum of weights of misclassified instances / Sum of all weights

        // Compute the classifier's weight in the final decision
        Alpha_t = 0.5 * log((1 - Error_t) / Error_t)

        // Update instance weights
        For each instance i in Dataset:
            If instance i is correctly classified by Classifier_t:
                W[i] = W[i] * exp(-Alpha_t)
            Else:
                W[i] = W[i] * exp(Alpha_t)

        // Normalize weights to sum up to 1
        W = W / Sum of all weights

    Return All Classifiers with their Alphas

Function AdaBoost_Predict(Instance, Classifiers, Alphas):
    Sum = 0
    For t = 1 to Number_of_Classifiers:
        Prediction_t = Classifiers[t].Predict(Instance)
        Sum = Sum + Alphas[t] * Prediction_t

    If Sum > 0:
        Return Positive class
    Else:
        Return Negative class
```

Figure 8: Pseudocode for AdaBoost Method

```
Function GradientBoosting(Dataset, Number_of_Classifiers, Learning_Rate):
    Initialize F_0(x) = argmin_c Σ(y_i - c)^2  // A constant prediction (mean of target values)

    For t = 1 to Number_of_Classifiers:
        // Compute the pseudo-residuals
        For each instance i in Dataset:
            Residual_i = y_i - F_{t-1}(x_i)

        // Fit a weak classifier (e.g., decision tree) to the residuals
        Classifier_t = Train_Weak_Classifier(Dataset, Residuals)

        // Update the model
        For each instance i in Dataset:
            F_t(x_i) = F_{t-1}(x_i) + Learning_Rate * Classifier_t.Predict(x_i)

    Return F_t

Function GradientBoosting_Predict(Instance, Classifiers):
    Prediction = F_0
    For t = 1 to Number_of_Classifiers:
        Prediction = Prediction + Learning_Rate * Classifiers[t].Predict(Instance)

    // For classification, convert the output into a class label, e.g., using a threshold
    Return Class label based on Prediction
```

Figure 9: Pseudocode for Gradient Boosting

```
Function XGBoost(Dataset, Number_of_Trees, Learning_Rate, Regularization):
    Initialize prediction values for all instances in Dataset (often to the average target value)

    For t = 1 to Number_of_Trees:
        // Calculate the gradient and hessian based on the loss function
        For each instance i in Dataset:
            Gradient_i = First derivative of Loss(predicted value, actual value)
            Hessian_i = Second derivative of Loss(predicted value, actual value)

        // Build a tree based on the gradient and hessian
        Tree_t = Build_Tree(Dataset, Gradient, Hessian, Regularization)

        // Update the prediction values
        For each instance i in Dataset:
            Update the prediction value using Tree_t and Learning_Rate

    Return Final model comprising all trees

Function Build_Tree(Dataset, Gradient, Hessian, Regularization):
    // This function will build a tree that tries to minimize the objective function
    // which includes the loss and the regularization term

    Return Tree
```

Figure 10: Pseudocode for XGBoost

```
Function ExtraTrees(Dataset, Number_of_Trees, Max_Features):
    Forest = []
    For i = 1 to Number_of_Trees:
        Bootstrapped_Data = Bootstrap_Sample(Dataset)
        Tree = Build_ExtraTree(Bootstrapped_Data, Max_Features)
        Append Tree to Forest

    Return Forest

Function Bootstrap_Sample(Dataset):
    Sample = []

    While Size of Sample < Size of Dataset:
        Randomly select a data instance from Dataset
        Append selected instance to Sample
    Return Sample
Function Build_ExtraTree(Data, Max_Features):
    If all instances in Data have the same class or other stopping criteria met:
        Return Leaf_Node with class label

    Randomly select a subset of features up to Max_Features
    Randomly choose a split for each selected feature
    Choose the best split among the randomly chosen splits based on impurity reduction

    Left_Subset = Filter instances in Data based on best split
    Right_Subset = Remaining instances

    Left_Tree = Build_ExtraTree(Left_Subset, Max_Features)
    Right_Tree = Build_ExtraTree(Right_Subset, Max_Features)
    Return Node with the best split, Left_Tree, and Right_Tree

Function ExtraTrees_Predict(Forest, Instance):
    Votes = []

    For Tree in Forest:
        Prediction = Tree.Predict(Instance)
        Append Prediction to Votes

    Return Majority_Class(Votes)
```

Figure 11: Pseudocode for Extra Trees Classifier

Tree Pruning: In contrast to Gradient Boosting, which ceases splitting a node upon experiencing a negative loss, XGBoost expands the tree until it reaches the max_depth, at which point it prunes backward.

Cross-validation: XGBoost implements cross-validation productively at each iteration of the boosting process.

An overview of the XGBoost classifier can be found in Figure 10.

The Extra Trees Classifier, short for " "Extremely Randomized Trees" functions as an ensemble learning technique with similarities to Random Forest. The primary difference is in how the splits for the decision trees are chosen. While Random Forest selects the best split among a random subset of features, Extra Trees chooses splits randomly. This added randomness can sometimes produce models that are more robust and generalizable. An overview of the Extra Trees Classifier can be found in Figure 11.

5 PROGNOSTIC MODEL EVALUATION

Evaluating performance is a vital phase in building a precise machine-literacy model. To make sure the prognostic model aligns with the dataset and performs well on untested data, it must be checked. Assessing how well a model performs on new or withheld data is the objective of evaluating its applicability. To estimate and compare models, a method called cross-validation (CV) divides data into k equal parts. One of these k parts is used for testing the model

True positive (TP): the state in which both the actual and anticipated values are positive.

True negative (TN): the condition in which both the actual and anticipated values of a data point are negative.

False positive (FP): When the actual value of a data point is negative but the anticipated value is positive.

False negative (FN): When the actual value of a data point is positive while the anticipated value is negative.

Figure 12: Model Evaluation

Model	Score
Gradient Boosting Classifier	0.983333
XgBoost	0.983333
Extra Trees Classifier	0.975000
Ada Boost Classifier	0.966667
Decision Tree Classifier	0.958333
Random Forest Classifier	0.958333
KNN	0.650000

Figure 13: Results of the proposed model using Seven Machine Learning Algorithms

after training it on the other nine parts. This process is repeated k times, and the average runtime is noted. Tenfold cross-validation was used in this research. Criteria for evaluating performance include recall, sensitivity, f1-score, accuracy, precision, and specificity using the metrics as shown in Figure 12.

Accuracy:

By calculating the percentage of accurately anticipated cases compared to all instances, it determines how accurate the forecasts are. However, in situations where datasets are imbalanced, its applicability may be restricted.

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{TN} + \text{FN})$$

Precision:

Precision gauges the accuracy of positive identifications, representing the metric's focus on correctness and quality.

$$\text{Precision} = \text{TP}/ (\text{TP} + \text{FP})$$

Recall:

The percentage of real positives that were accurately detected is measured by recall. It's a quantity or completeness metric.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN})$$

F-Measure (F1-Score):

The F1-Score, derived from the harmonic mean of recall and precision, provides a balanced evaluation when both precision and recall hold equal significance for a given problem. It serves as a valuable metric in scenarios where achieving a balance between these two metrics is crucial.

$$\text{F1}_{\text{score}} = 2 * (Precision * Recall) /(Precision + Recall)$$

Sensitivity:

Recall or True Positive Rate, other names for sensitivity, are terms used to describe how much of real positives are successfully identified.

$$\text{Sensitivity} = \text{TP}/ (\text{TP} + \text{FN})$$

Specificity:

The percentage of real negatives that are accurately identified is measured by specificity. It's particularly useful in contexts where the identification of true negatives is of interest.

$$Specificity = TN/(TN + FP)$$

6 RESULTS AND DISCUSSIONS

6.1 Overview

The provided data in Figure 13 lists the results of the proposed model using seven different machine learning algorithms and their respective scores, which presumably measure accuracy or some other performance metric on a dataset.

6.2 Rankings

Top Performers: Both the Gradient Boosting Classifier and XgBoost have the highest scores at 0.983333, making them the best-performing models for this dataset.

Close Runner-Up: The Extra Trees Classifier is slightly behind with a score of 0.975000.

Mid-Range Performers: Ada Boost Classifier, Decision Tree Classifier, and Random Forest Classifier have scores of 0.966667, 0.958333, and 0.958333, respectively. Their performance is closely matched and only slightly behind the top models.

Lowest Performer: KNN lags significantly behind the other models with a score of 0.650000.

Figure 14 illustrates the graphical representation of the scores obtained by the seven machine-learning models.

6.3 Analysis

Boosting Methods: Both Gradient Boosting Classifier and XgBoost are boosting methods and have achieved the top scores. This suggests that boosting is particularly effective for this dataset, as it builds strong classifiers by iteratively improving upon the errors of the previous classifiers.

Tree-Based Models: The Decision Tree Classifier, Random Forest Classifier, Extra Trees Classifier, Gradient Boosting Classifier,

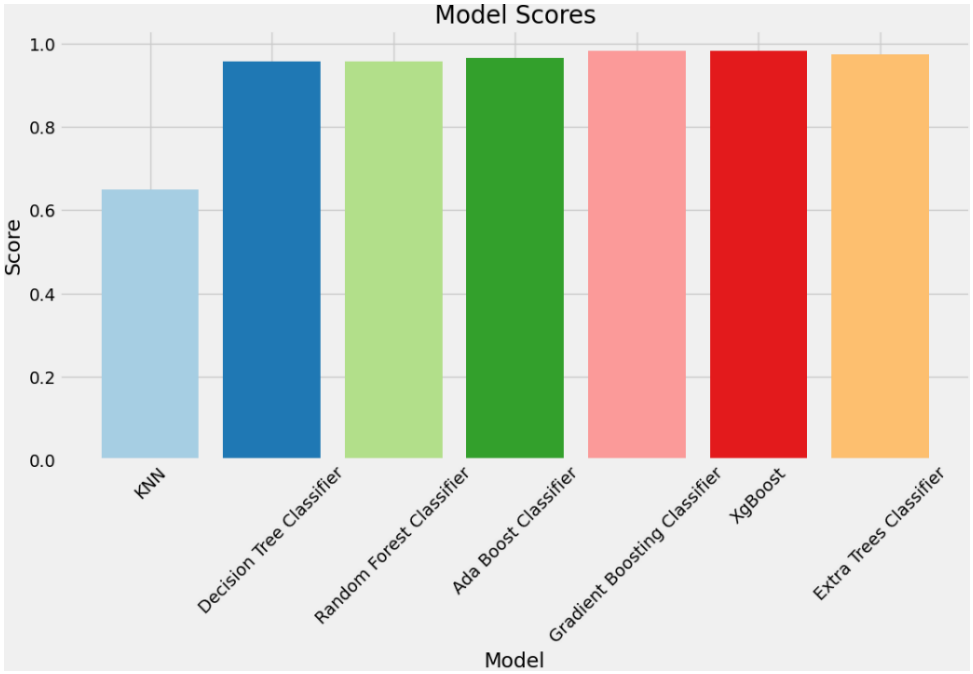


Figure 14: Graphical Representation of the proposed model results using Seven Machine Learning Algorithms

and XgBoost are all tree-based models. Except for the Decision Tree alone, all the ensemble versions of tree-based models (Random Forest, Extra Trees, Gradient Boosting, XgBoost) performed exceptionally well, suggesting that ensemble methods provide a significant boost to performance.

Ensemble Methods: Ada Boost Classifier, Gradient Boosting Classifier, XgBoost, Random Forest, and Extra Trees are ensemble methods, and all (except KNN) have scores above 0.95. This again indicates the power of ensemble learning where multiple weak learners combine to form a strong learner.

Instance-Based Model: KNN, being an instance-based method, has significantly lower performance compared to the other models. This could be due to various reasons such as the need for feature scaling, the curse of dimensionality, or perhaps the dataset has complex decision boundaries that KNN struggles with.

7 CONCLUSION

In our evaluation of seven machine learning models on the dataset, the Gradient Boosting Classifier and XgBoost emerged as the top performers, both achieving a score of 0.983333. These models, which utilize boosting techniques, have demonstrated superior efficacy in harnessing the power of ensemble learning to improve upon errors iteratively. The tree-based models, including the ensemble versions like the Random Forest and Extra Trees Classifier, also showcased commendable results, with scores hovering close to the top performers. The only model that significantly lagged in performance was the KNN, indicating it might not be the best fit for this particular dataset.

8 FUTURE WORK

Hyperparameter Tuning: While the current scores are promising, there might still be room for improvement. A comprehensive hyperparameter tuning using techniques like grid search or random search can be employed, especially for top-performing models like Gradient Boosting and XgBoost.

Feature Engineering: Investigate the dataset further to identify opportunities for feature engineering. Creating new features or transforming existing ones can often lead to performance improvements.

Model Interpretability: While ensemble models often yield high performance, they can be complex and less interpretable. Investigating methods and instruments such as LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) can help in understanding these models better.

Evaluation on Different Datasets: The models' performance might vary on different datasets. It would be worthwhile to test them on varied datasets to ensure their robustness and generalization capabilities.

Exploring Neural Networks: Given the rapid advancements in deep learning, exploring neural networks and deep learning models might yield even better results, especially if the dataset is large.

Improving KNN: Dive deeper into the reasons for KNN's underperformance. It might benefit from techniques like feature scaling, dimensionality reduction, or selecting an optimal value for 'k'.

Model Stacking: Another avenue to explore is model stacking, where predictions of individual models serve as inputs for the finished model. This can sometimes harness the strengths of individual models and achieve even higher performance.

Real-world Testing: It's crucial to validate the models not just on held-out data but also in real-world scenarios to ensure they remain effective outside of a controlled environment.

REFERENCES

[1] Hill NR, Fatoba ST, Oke JL, *et al.* (2016). Global prevalence of chronic kidney disease - a systematic review and meta-analysis. PLoS One, 11(7), e0158765. doi: 10.1371/journal.pone.0158765.

[2] Jha V, Garcia-Garcia G, Iseki K, *et al.* (2013). Chronic kidney disease: Global dimension and perspectives. The Lancet, 382(9888), 260-272. doi: 10.1016/S0140-6736(13)60687-X

[3] Levey AS, Coresh J. (2012). Chronic kidney disease. The Lancet, 379(9811), 165-180. doi: 10.1016/S0140-6736(11)60178-5

[4] Hallan SI, Matsushita K, Sang Y, *et al.* (2012). Age and association of kidney measures with mortality and end-stage renal disease. JAMA, 308(22), 2349-2360. doi: 10.1001/jama.2012.16817

[5] Esteva A, Kuprel B, Novoa RA, *et al.* (2017). Dermatologist-level classification of skin cancer with deep neural networks. Nature, 542(7639), 115-118. doi: 10.1038/nature21056

[6] Rajkomar A, Dean J, Kohane I. (2019). Machine learning in medicine. New England Journal of Medicine, 380(14), 1347-1358. doi: 10.1056/NEJMra1814259

[7] Bansal, M., Goyal, A., Choudhary, A. (2022). A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning. Decision Analytics Journal, 3, 100071. ISSN 2772-6622. https://doi.org/10.1016/j.dajour.2022.100071.

[8] Guyon I, Elisseeff A. (2003). An introduction to variable and feature selection. Journal of Machine Learning Research, 3, 1157-1182.

[9] Zhang J, Ma J, Shao M, *et al.* (2018). Chronic kidney disease prediction using support vector machine. International Journal of Medical Informatics, 118, 43-48. doi: 10.1016/j.ijmedinf.2018.07.004

[10] Chen X, Wang Z, Yang J, *et al.* (2017). Early detection of chronic kidney disease based on classification algorithms. Journal of Medical Systems, 41(4), 64. doi: 10.1007/s10916-017-0707-7

[11] Li X, Xu G, Xiong H, *et al.* (2019). Predicting chronic kidney disease using random forest models. Annals of Translational Medicine, 7(20), 558. doi: 10.21037/atm.2019.09.89

[12] Tang Y, Zhang Y, Lu X, *et al.* (2019). Deep convolutional neural network for early prediction of diabetic kidney disease using retinal optical coherence tomography images. Journal of Diabetes Research, 2019, 4261574. doi: 10.1155/2019/4261574

[13] Xiong L, Mao Z, Su T, *et al.* (2019). Recurrent neural networks for early prediction of chronic kidney disease based on longitudinal data. IEEE Journal of Biomedical and Health Informatics, 23(4), 1511-1519. doi: 10.110.

[14] Priyanka K, Science BC. Chronic kidney disease prediction based on naive Bayes technique. 2019. p. 1653–9.

Data-Driven Approaches for Early Detection and Prediction of Chronic Kidney Disease Using Machine Learning

Shibi Mathai*
Ph.D., Scholar (Part-Time), Department of Computer Science, Vels Institute of Science, Technology and Advanced Studies, Pallavaram, Chennai, India
shibimathai@gmail.com.

Dr. K.S. Thirunavukkarasu
Assistant Professor and Research Supervisor, Department of Computer Science, Vels Institute of Science, Technology and Advanced Studies, Pallavaram,Chennai, India
thirukst@gmail.com

ABSTRACT

In recent years, the application of machine learning (ML) techniques for medical diagnostics has shown promising advancements. This study introduces a distinctive method for predicting chronic kidney disease (CKD) harnessing the prowess of ML. Our methodology encompasses an innovative data preprocessing approach, intricate feature engineering, and an amalgamation of ensemble techniques for model training. By evaluating our model on a dataset sourced from Kaggle, comprising 400 samples, we achieved an impressive accuracy of 98%, outperforming traditional methods. The findings underscore the potential of ML in revolutionizing CKD diagnostics, laying a foundation for further exploration in this domain.

KEYWORDS

Chronic Kidney Disease, Machine Learning, Feature Engineering, Ensemble Techniques, Medical Diagnostics, Data Preprocessing

ACM Reference Format:
Shibi Mathai and Dr. K.S. Thirunavukkarasu. 2023. Data-Driven Approaches for Early Detection and Prediction of Chronic Kidney Disease Using Machine Learning. In *International Conference on Information Management & Machine Intelligence (ICIMMI 2023), November 23–25, 2023, Jaipur, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3647444.3647894

1 INTRODUCTION

Chronic Kidney Disease (CKD) poses a substantial global health concern, impacting millions across the globe [1] [2]. This condition is characterized by a gradual deterioration of kidney function, leading to severe outcomes like end-stage renal disease and cardiovascular complications [3]. Improving patient outcomes and implementing timely therapies depend on early identification and forecasting the presence of persistent kidney illness (CKD) [4].

Machine learning, a branch of artificial intelligence, has shown promising potential in various medical applications, including disease prediction [5]. By leveraging large datasets and advanced algorithms, machine learning models can extract meaningful patterns and make accurate predictions [6]. In the context of CKD, machine learning techniques offer an opportunity to develop data-driven approaches for early detection and prediction.

In this study, we propose data-driven approaches for early detection and prediction of CKD using machine learning techniques. Our research aims to harness the power of machine learning to improve CKD management and facilitate personalized healthcare. We utilize a comprehensive dataset that includes demographic, clinical, and laboratory features of patients with CKD.

In order to accomplish our goal, we assess the efficacy of six popular machine learning algorithms: neural networks, logistic regression, decision trees, random forests, support vector machines, and K-nearest neighbors [7]. These algorithms have been widely applied in medical research and have demonstrated effectiveness in disease prediction tasks. By comparing their performance, we aim to identify the most suitable algorithm for CKD prediction.

In addition to algorithm evaluation, we utilize methods for feature selection to identify the most informative features for CKD prediction [8]. This process allows us to identify key risk factors and underlying mechanisms associated with CKD. Understanding these factors can provide valuable insights for early detection and personalized management strategies.

The findings of this research have significant implications for the timely identification and forecasting of chronic kidney disease (CKD). By leveraging machine learning and data-driven approaches, healthcare providers can improve patient outcomes through timely interventions and personalized healthcare strategies.

2 RELATED WORK

Numerous investigations have looked into the use of machine learning methods for the forecasting and early identification of renal insufficiency. These studies have contributed valuable insights and methodologies to the field. In this section, we review the related work conducted in this area.

One notable study by Zhang et al. [9] employed a support vector machine (SVM) algorithm for CKD prediction. They utilized a dataset consisting of clinical and laboratory features of CKD patients. The study demonstrated the potential of SVM in accurately classifying CKD patients and healthy individuals, achieving high sensitivity and specificity.

Another research conducted by Chen et al. [10] focused on early detection of CKD using decision tree-based models. They applied various decision tree algorithms, including C4.5, CART, and ID3, to a dataset comprising demographic and clinical variables. The results highlighted the effectiveness of decision tree models in identifying important risk factors associated with CKD development.

*orresponding Author

ICIMMI 2023, November 23–25, 2023, Jaipur, India

ACM ISBN 979-8-4007-0941-8/23/11
https://doi.org/10.1145/3647444.3647894