Liver Disease Analysis and Prediction

A Project Report Submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfilment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Computer and Communication Engineering

by

Snigdha Shambhavi Jha, Anusha Arra, Jinay Patel

210953082, 210953092, 210953236

Under the guidance of

Dr. Chethan Sharma
Assistant Professor- Senior Scale
Department of I&CT
Manipal Institute of Technology
Manipal, Karnataka, India

Dr. Manjula Shenoy
Professor
Department of I&CT
Manipal Institute of Technology
Manipal, Karnataka, India

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Unit of MAHE, Manipal

November 2023

INDEX

ABSTRACT

Diseases of the liver are many in number, and therefore they pose a significant health challenge all around the globe, with early detection being crucial for effective intervention and patient care.

This Data Mining and Predictive Analysis Lab project, made by one of the teams from V Sem CCE-A1 batch, focuses on the application of advanced data mining techniques to predict, and analyse liver diseases based on a patient dataset.

Predictive modelling using data mining approaches offers a promising avenue for early detection and prognosis of liver diseases. Leveraging machine learning algorithms and statistical techniques on patient data can provide valuable insights into the risk factors, patterns, and potential indicators of liver diseases.

This Data Mining Lab project aims to explore and implement predictive models that can assist in the timely identification and management of liver diseases, ultimately contributing to improved patient outcomes and healthcare decision-making.

INTRODUCTION

The liver plays a vital role in the body's metabolism, and disorders affecting it can lead to severe health complications.

Liver disease prediction through data mining techniques is a critical area of research in the field of healthcare.

Through the utilization of machine learning and genetic algorithms, and artificial neural networks, along with statistical methodologies, we as a team aim to develop predictive analysis models that can help point out and identify key risk factors, patterns, and potential indicators associated with liver diseases based on inputs like liver enzymes, for example.

The project involves preprocessing and analysing a diverse dataset, implementing, and fine-tuning predictive models, and evaluating their performance. The insights gained from this project have the potential to enhance early diagnosis, treatment planning, and overall healthcare management for individuals at risk of or currently experiencing liver diseases.

LITERATURE SURVEY

LITERATURE REVIEW: LIVER DISEASE PREDICTION USING DATA MINING, MACHINE LEARNING AND ANN'S (GENETIC ALGORITHMS)

[1] Using a range of decision tree techniques, including J48, REP Tree, Random Tree, Decision Stump, LMT, Random Forest, and Hoeffding Tree, this study attempts to forecast liver disorders. The main goal is to thoroughly compare different decision tree algorithms in order to identify the best one for predicting liver disease. This strategy is important because it finds the best decision tree algorithm for predicting liver illnesses, which offers insightful information for medical applications. However, difficulties are recognised, including the difficulty of interpreting sophisticated machine learning algorithms and the restricted applicability of the findings to a wide range of demographics. The algorithms' performance examination shows that the accuracies vary, with Decision Stump having the highest accuracy at 70.67%. Other algorithms that perform well include Hoeffding Tree (69.75%), Random Tree (69.47%), LMT (69.30%), and J48 (65.49%). The conclusion drawn from this analysis is that the Decision Tree algorithm outperforms others in predicting liver diseases, contributing to the understanding of optimal machine learning techniques for healthcare applications.

[2] This research investigates liver disease prediction through a comparative study of six supervised machine learning algorithms: Logistic Regression (LR), K-Nearest Neighbors (KNN), Decision Tree (DT), Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest (RF). Via the application of these techniques, their performance is evaluated from various perspectives. The primary advantage of the same lies in determining the most effective supervised machine learning algorithm for predicting liver diseases. However, the research admits to potential disadvantages of overfitting, wherein models may perform exceptionally well on training data but exhibit reduced accuracy on new, untested data, potentially leading to unreliable predictions in real-world scenarios. Analysis of the algorithms' performances reveals that Logistic Regression achieved the highest accuracy at 53%, with Naive Bayes recording the lowest accuracy at 75%. Logistic Regression also outperformed in terms of precision (91%) and F1 measure (83%), while SVM exhibited the highest sensitivity (88%). Decision Tree excelled in specificity with a value of 48%. In conclusion, based on the outlined measurement criteria, Logistic Regression emerges as the superior classification technique for predicting chronic liver disease.

[3] The methodology involves a sequence of steps, including normalization, feature selection, the application of various classification algorithms, and the evaluation of validation metrics to determine accuracy results. The advantages of this approach lie in the ability to compare different algorithms, each with its specific advantages and disadvantages, enabling healthcare professionals to choose the most suitable algorithm for their patient group or healthcare environment. However, the research acknowledges a limitation: while machine learning

models can identify data correlations, they may not reveal causal linkages crucial for understanding and treating diseases. The performance analysis showcases the accuracy results of different classification algorithms, including J48, MLP, SVM, Random Forest, and BayesNet, under both Greedy Stepwise and Particle Swarm Optimization (PSO) feature selection frameworks. Notably, the PSO approach yields superior results, with BayesNet and J48 Classification demonstrating higher performance compared to other algorithms. The research concludes that, when used within the PSO feature selection framework, BayesNet and J48 Classification exhibit superior performance in predicting liver disease using the Indian Liver Patient Dataset.

[4] This research addresses the prediction of liver disease utilizing machine learning algorithms, employing K-Nearest Neighbour (KNN), Decision Tree (DT), and Adaptive Neuro-Fuzzy Inference System (ANFIS). The methodology involves data collection, preprocessing, and the application of the classification algorithms after splitting the data into training and testing sets. The advantages of each algorithm are highlighted: KNN offers ease of implementation and handles non-linear relationships; DT captures complex boundaries and identifies important features; ANFIS models complex, non-linear relationships by combining neural networks and fuzzy logic. However, each algorithm has its drawbacks, such as KNN's computational expense with large datasets and sensitivity to the "K" parameter, DT's potential for overfitting, instability, and ANFIS's interpretability challenges and computational expense. Performance evaluation reveals accuracy rates of 66.6% for KNN, 89.09% for DT, and 98% for ANFIS. In terms of sensitivity, ANFIS outperforms KNN with 97.4% compared to 76.61%. Precision is highest for ANFIS at 100%, whereas KNN scores 76.2%. Considering specificity, KNN achieves 42%, while ANFIS reaches 100%. In conclusion, ANFIS demonstrates the highest accuracy among the three techniques, emphasizing its effectiveness in liver disease prediction.

[5] This survey focuses on the prediction and analysis of liver disorder diseases through the application of data mining techniques. The research employs Association, Clustering, and Classification methods, specifically Naive Bayes, Support Vector Machine (SVM), and C4.5 Decision Tree. The methodology involves preprocessing the dataset, selecting relevant features, and partitioning the data for classification. The advantages of each technique are highlighted, with Naive Bayes offering simplicity, speed, and high probability classification, SVM providing fast estimation and better prediction results with fewer parameters, and C4.5 Decision Tree requiring less model build and search time, utilizing minimal memory, and delivering accurate results. However, the research acknowledges the disadvantages of each technique, such as Naive Bayes' need for a large number of records, SVM's computational expense, and C4.5 Decision Tree's susceptibility to overfitting and potential inclusion of empty or redundant branches. The study concludes by emphasizing the significance of data mining, particularly clustering, classification, and association, in diagnosing diseases. Furthermore, it suggests that future work can explore hybrid approaches to enhance the performance accuracy of liver disorder disease prediction.

[6] This study focuses on the rule optimization of the Boosted C5.0 classification algorithm using Genetic Algorithm for liver disease prediction. MDM (Medical Data Mining) concepts are employed to enhance the regular C5.0 classification algorithm through boosting, where Genetic Algorithm is incorporated in three key sections: Selection, Crossover, and Mutation. The advantages of this approach include improved accuracy, reduced overfitting, and

enhanced handling of complex datasets. The boosted model adapts to imbalanced data, selects relevant features, and creates a robust, interpretable ensemble model, making it powerful for various classification tasks. However, it comes with certain disadvantages, such as computational intensity, especially for large datasets, potential overfitting with improper parameter tuning, and reduced interpretability compared to standalone C5.0 trees. The performance comparison between Genetic Algorithm (GA) and Boosted C5.0 showcases trade-offs in specificity, sensitivity, precision, false positive rate, false discovery rate, F1 score, and overall accuracy. GA, after implementation, generated 24 rules compared to the initial 92, indicating an optimized method for rule generation in the context of liver disease prediction.

[7] This research addresses the use of intelligent techniques for liver disease prediction, employing Random Forest (RF), Multilayer Perceptron (MLP) model, k Nearest Neighbour (kNN), and Support Vector Machine (SVM). The significant advantages of these methods include not only improved diagnosis by identifying precise genetic flaws associated with liver disease but also the potential discovery of novel drug targets for more effective and targeted therapies. Despite these promising outcomes, the research conscientiously acknowledges ethical concerns, such as the risk of unfair treatment based on genetic information and privacy issues. The comparative performance analysis reveals that the SVM approach attains the highest accuracy at 74%, followed by Random Forest at 69%, MLP at 68%, and kNN at 67%. While acknowledging the potential for further advancements with additional methodologies in the future, the findings underscore the current value of these techniques in enhancing the accuracy of liver disease prediction.

[8] This study focuses on leveraging Naïve Bayes and Support Vector Machine (SVM) algorithms for the prediction of liver diseases. The Naïve Bayes classifier, known for its applicability even when study assumptions aren't strictly met and its efficiency with minimal training data, is utilized alongside the Support Vector Machine. While the Naïve Bayes approach achieves an accuracy of 61.28%, the SVM method outperforms with a significantly higher accuracy of 79.66%. Despite the computational time disadvantage associated with SVM, the study concludes that it is the preferred method for classifying liver diseases due to its superior accuracy. This research emphasizes the practicality and effectiveness of SVM in the specific context of liver disease prediction.

[9] This research addresses the diagnosis of liver disease by employing the Support Vector Machine (SVM) tool optimized through the Crow Search Algorithm (CSA). The methodology involves the application of SVM, optimization using the Sequential Minimal Optimization (SMO) Algorithm, kernel functions, and alpha values, coupled with the innovative use of the Crow Search Algorithm for metaheuristic optimization. The advantage of incorporating the CSA lies in providing a new perspective and generating diverse outcomes compared to using the SVM algorithm alone. The CSA-SVM hybrid approach achieves an impressive accuracy of 99.49%, outperforming other methods with a singular focus on predicting liver disease. This study highlights the efficacy of combining SVM with metaheuristic optimization techniques, specifically the Crow Search Algorithm, for enhanced accuracy in liver disease diagnosis.

[10] This research delves into the analysis of liver disease prediction through various machine learning methods, including Classification, Random Forest, Support Vector Machine

(SVM), Multilayer Perceptron (MLP), Bayesian Network, and J48. The methodology encompasses a classification of selection algorithms into heuristic search, complete search, meta-heuristic methods, and artificial neural network (ANN) methods. While acknowledging the disadvantages, particularly the substantial gap between existing and proposed accuracy data models, the study reports promising performance results for the proposed methods. Notably, J48 achieves an accuracy of 95.04%, Random Forest at 80.22%, Bayesnet at 90.33%, MLP at 77.54%, and SVM at 73.44%. The conclusion drawn suggests the versatility of this approach, indicating its potential application in other disease diagnosis methods for accurate analysis beyond liver disease prediction.

METHODOLOGY

The dataset that we as a team have chosen is the ILPD (Indian Liver Patient Dataset). It consists of the records of 583 patients, in which 416 patients have been diagnosed with liver disease, and there are 167 without liver disease. It has a wide range of features and attributes which enable us to analyse the data and make appropriate predictions, which are as follows:

Age: Age of the Patient

Gender: Sex of the Patient

TB: Total Bilirubin Levels

DB: Direct Bilirubin Levels

AlkPhos: Alkaline Phosphatase Levels

Sgpt: Alamine Aminotransferase Levels

Sgot: Aspartate Aminotransferase Levels

TP: Total Proteins Levels

ALB: Albumin Levels

A/G Ratio: Albumin and Globulin Ratio

Selector: field used to split the data into two sets (patient with liver disease, or no disease) [target variable used for prediction]

The data set mentioned can be found on the UC Irvine Machine Learning Repository

(https://archive.ics.uci.edu/dataset/225/ilpd+indian+liver+patient+dataset)

The dataset contains three types of features: Continuous, Integer and Binary

Continuous features: [‘TB’, ‘DB’, ‘TP’, ‘ALB’, ‘A/G Ratio’]

Integer features: [‘Age’,’Alkphos’, ‘Sgpt’, ‘Sgot’]

Binary features: [‘Gender’, ‘Selector’]

DATA PREPROCESSING

Data Loading and Inspection: After loading the dataset into a pandas DataFrame, the column names were changed to reflect the changes. Each column's missing values were found using the isnull() technique.

```
col_names = ['Age','Gender','TB_total_bilirubin', 'DB_Direct_Bilirubin','Alkphos_Alkaline_Phosphotase', 'Sgpt_Alamine_Aminotransferase',
       'Sgot_Aspartate_Aminotransferase', 'TP_Total_Protiens', 'ALB_Albumin', 'A/G_Ratio','Selector Field']
len(col_names)
```

```
11
```

```
data.columns = col_names
```

```
data.head()
```

	Age	Gender	TB_total_bilirubin	DB_Direct_Bilirubin	Alkphos_Alkaline_Phosphotase	Sgpt_Alamine_Aminotransferase	Sgot_Aspartate_Aminotransferase	TP_Total_Protiens
0	62	Male	10.9	5.5	699	64	100	7.5
1	62	Male	7.3	4.1	490	60	68	7.0
2	58	Male	1.0	0.4	182	14	20	6.8
3	72	Male	3.9	2.0	195	27	59	7.3
4	46	Male	1.8	0.7	208	19	14	7.6

Missing Value Imputation: The column mean was used to impute missing values from the A/G_Ratio column.

```
data.isnull().sum() #check for missing values in each column and returns the count in each column
```

```
Age                           0
Gender                        0
TB_total_bilirubin            0
DB_Direct_Bilirubin           0
Alkphos_Alkaline_Phosphotase  0
Sgpt_Alamine_Aminotransferase 0
Sgot_Aspartate_Aminotransferase 0
TP_Total_Protiens             0
ALB_Albumin                   0
A/G_Ratio                     4
Selector Field                0
dtype: int64
```

```
data['A/G_Ratio'].fillna(round(data['A/G_Ratio'].mean(),2),inplace=True)
```

```
data.isnull().sum()
```

```
Age                           0
Gender                        0
TB_total_bilirubin            0
DB_Direct_Bilirubin           0
Alkphos_Alkaline_Phosphotase  0
Sgpt_Alamine_Aminotransferase 0
Sgot_Aspartate_Aminotransferase 0
TP_Total_Protiens             0
ALB_Albumin                   0
A/G_Ratio                     0
Selector Field                0
dtype: int64
```

Data Encoding: The map() function has been used to numerically encode categorical information, such as gender and selector field. Gender is indicated by 0 for "Male" and 1 for "Female".

"No Disease" was denoted by 0 in the Selector Field and "Disease Present" by 1.

```
data['Selector Field'] = data['Selector Field'].map(lambda x:0 if x==2 else 1)  # 0-no disease, 1-disease
```

```
data['Gender'] = data['Gender'].map(lambda x:0 if x=="Male" else 1)
```

```
value_table = pd.crosstab(data['Gender'],data['Selector Field'])
value_table.columns = ["No Disease","Disease Present"]
value_table.index = ["Male","Female"]
print(value_table)
```

```
        No Disease  Disease Present
Male           117              324
Female          50               91
```

Exploratory Data Analysis: To look at the association between gender and selector field, a cross-tabulation was done. This revealed information about the disease's prevalence in each gender.

Data Splitting: The train_test_split() function from scikit-learn was used to divide the dataset into training and testing sets. Thirty percent of the data were in the test set, while the remaining seventy percent were utilised for training.

```
#Split

X = data.drop('Selector Field',axis=1)
y = data['Selector Field']
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_Scaled = scaler.fit_transform(X)
```

```
Xtrain, Xtest, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(Xtrain)
X_test = scaler.transform(Xtest)
```

Feature Scaling: Scikit-learn's StandardScaler() was used to standardise the training and test data. This guaranteed a mean of 0 and a standard deviation of 1 for every feature.

The dataset was suitably prepared for later machine learning tasks by the data preparation approach described above. The data was divided into training and testing sets, missing values were imputed, categorical variables were encoded, and feature scaling was used. By taking these precautions, the data was guaranteed to be pristine, reliable, and appropriate for model testing and training.

IMPLEMENTATION

We have chosen to implement 4 ML algorithms - SVM, Logistic Regression, Random Forest and KNN (K-Nearest Neighbour). We have compared the results by adding a K-Fold component to them.

Additionally, we have also implemented a genetic algorithm to choose the best features needed for the prediction and designed an ANN for the same.

K-FOLD CROSS VALIDATION

One method for evaluating a machine learning model's performance and capacity for generalisation is K-fold cross-validation. The process involves partitioning the dataset into k folds of equal size. The remaining k-1 folds are used as the training set and a new fold is used as the test set for each of the k training and evaluation cycles of the model. In the course of the cross-validation process, this helps to guarantee that each data point is used for both training and testing.

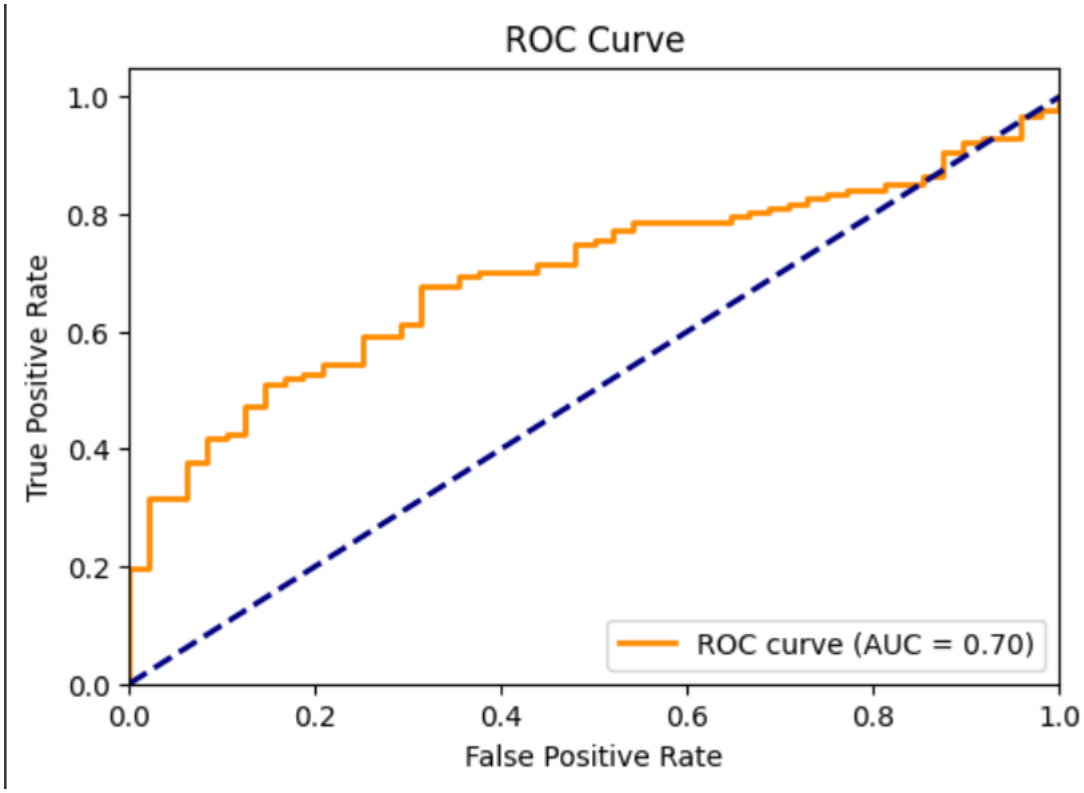
To give a more thorough assessment of the model's performance, the performance measures (such as accuracy, precision, or mean squared error) are averaged over the k iterations. As it makes the most of the data available for both training and testing, K-fold cross-validation is particularly helpful when there is a limited amount of data available. It also aids in gaining a more accurate estimate of the model's performance on unseen data. The values of 5 and 10 are often used for k, however the exact value will depend on the size and features of the dataset.

SUPPORT VECTOR MACHINE

For problems involving regression and classification, Support Vector Machines (SVM) represent a potent class of supervised learning algorithms. SVM's basic idea is to locate the hyperplane in a high-dimensional space that optimally divides data points belonging to various classes. The objective of classification is to determine a decision boundary with the largest margin, which is the separation between each class's closest data points and the hyperplane. SVMs are very good at addressing non-linear connections since they translate the input data into a higher-dimensional space (where a linear separation would be possible) using a kernel method. Finding support vectors—the data points that are closest to the decision boundary—and building a hyperplane based on those vectors is the main concept.

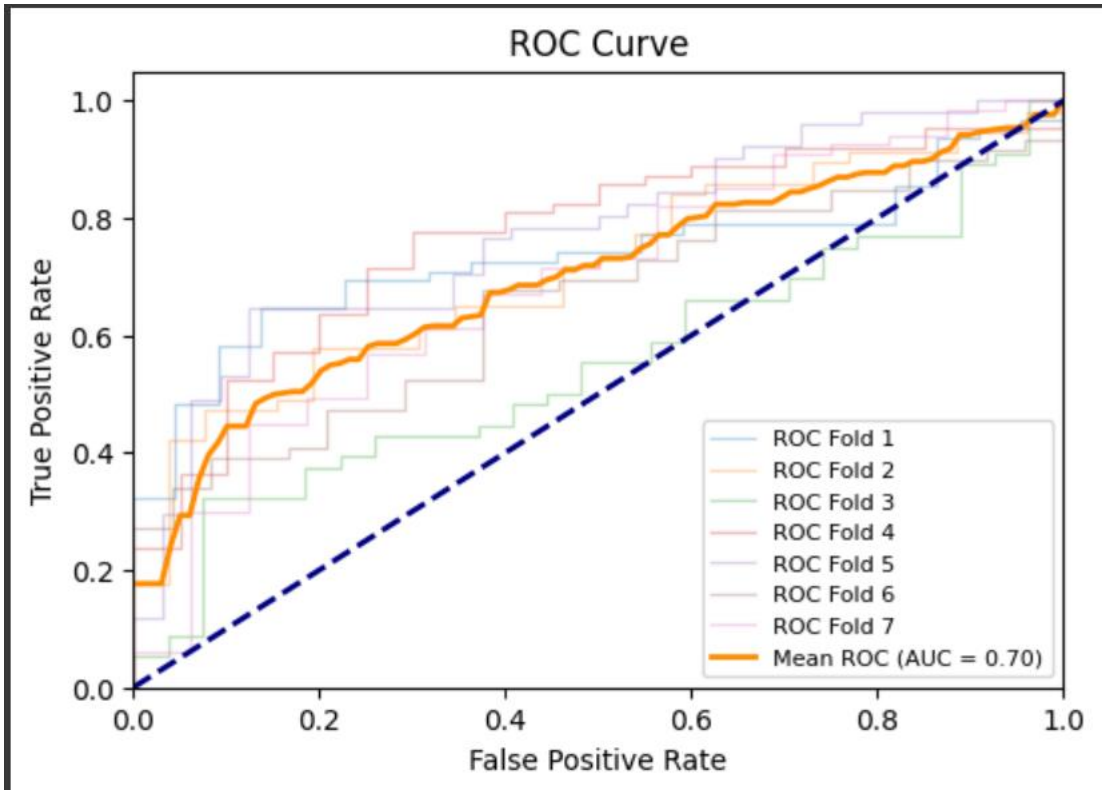
SVMs are renowned for their resilience, adaptability, and strong generalisation to fresh, untested data. SVMs are useful for a variety of tasks, including text classification, picture classification, and bioinformatics. The optimization procedure entails reducing a cost function that penalises misclassifications while maximising the margin.

```
from sklearn import svm
from sklearn.svm import SVC

svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)

accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {accuracy_svm:.2f}")
print(classification_report(y_test, y_pred_svm))
```



Using K-Fold,

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
k = 7
kf = KFold(n_splits=k, shuffle=True, random_state=1)
svm_model = SVC(kernel='linear')
scores = cross_val_score(svm_model, X_Scaled, y, cv=kf, scoring='accuracy')
print("SVM Cross-Validation Scores:")
print(scores)
print(f"Mean Accuracy: {scores.mean():.2f}")
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Fold 1
ROC Fold 2
ROC Fold 3
ROC Fold 4
ROC Fold 5
ROC Fold 6
ROC Fold 7
Mean ROC (AUC = 0.70)

LOGISTIC REGRESSION

Logistic Regression is a popular statistical technique for binary classification tasks—where the objective is to estimate the likelihood that an instance belongs to a specific class—is logistic regression. The basic concept is to use the logistic function to represent the connection between the independent factors and the log-odds of the dependent variable falling into a specific class. Every real-valued integer is mapped by the logistic function, also called the sigmoid function, to a value between 0 and 1, which is read as a probability. The method modifies the input feature weights during training in order to maximise the probability of the observed results. In order to categorise cases into one of the two groups, a decision threshold is applied to the probabilities that the logistic regression model produces.

Logistic regression is a useful tool in scenarios where it is expected that there is a roughly linear connection between the target variable and the characteristics, both in terms of log-odds and computing efficiency.

In this case, the maximum iterations have been set to 1000.

```
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
y_pred_logistic = logistic_model.predict(X_test)

accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print(f"Logistic Regression Accuracy: {accuracy_logistic:.2f}")
print(classification_report(y_test, y_pred_logistic))
```

Using K-Fold,

```
logistic2_model = LogisticRegression(max_iter=1000)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(logistic2_model, X_Scaled, y, cv=kfold, scoring='accuracy')
for fold, score in enumerate(scores, start=1):
    print(f"Fold {fold} Accuracy: {score:.2f}")
mean_accuracy = scores.mean()
print(f"Mean Accuracy: {mean_accuracy:.2f}")
```

RANDOM FOREST ALGORITHM

An ensemble learning method called Random Forest is applied to both regression and classification problems. Building a large number of decision trees during the training phase and combining their predictions during the testing or inference phase is the fundamental notion behind a Random Forest.

A predetermined number of decision trees are constructed during the training phase, usually by employing a method known as bagging (Bootstrap Aggregating). In order to generate several subsets for training each tree, bagging entails repeatedly sampling with replacement from the training data. To further diversify the trees, at each split, a random subset of features is taken into consideration for each tree.

Each Random Forest tree makes its own independent predictions when it comes to classification or regression, and the final result is decided by an average in the case of regression and a majority vote in the case of classification. Comparing this ensemble method to individual decision trees, it is generally less prone to overfitting and more resilient.

Scikit-learn Random Forests balance computational efficiency and accuracy by defining the number of decision trees in the ensemble using 'n_estimators'. Reproducibility is ensured by initialising the random number generator with the 'random_state' option. Although it increases computational cost, a greater 'n_estimators' can improve model robustness. 'random_state' gives the random number generator a particular seed, which ensures consistency in results between runs.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

clf = RandomForestClassifier(n_estimators=100, random_state=0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy for random forest: {accuracy}')
print(classification_report(y_test,y_pred))
print(metrics.confusion_matrix(y_test,y_pred))
```

Using K-Fold,

```
knn2_model = KNeighborsClassifier(n_neighbors=10)
kfold = KFold(n_splits=7, shuffle=True, random_state=42)
scores = cross_val_score(knn2_model, X_Scaled, y, cv=kfold, scoring='accuracy')
for fold, score in enumerate(scores, start=1):
    print(f"Fold {fold} Accuracy: {score:.2f}")
mean_accuracy = scores.mean()
print(f"Mean Accuracy: {mean_accuracy:.2f}")
```

K-NEAREST NEIGHBOUR

A straightforward and understandable machine learning approach for classification and regression problems is called K-Nearest Neighbors, or k-NN. The majority class (for classification) or the average (for regression) of a new data point's k nearest neighbours in the feature space is the basis for the prediction in k-NN. The proximity of data points is determined by the distance metric, which is typically Euclidean distance, and the variable "k" denotes the number of neighbours to take into account. The algorithm finds the k closest neighbours during prediction and gives the new data point the label or value that most closely matches theirs. For best results, the algorithm's susceptibility to noise and model complexity are affected by the choice of k, which calls for careful tweaking. Despite being simple, k-NN might have issues with high-dimensional data and high computational costs.

```
knn_model = KNeighborsClassifier(n_neighbors=11)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)

accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {accuracy_knn:.2f}")
print(classification_report(y_test, y_pred_knn))
```

Using K-Fold,

```
k = 7
kf = KFold(n_splits=k, shuffle=True, random_state=42)

rf_model = RandomForestClassifier(n_estimators=150, random_state=0)

X_df = pd.DataFrame(X_Scaled, columns=X.columns)
y_df = pd.Series(y, name='Target')

classification_reports = []
accuracies = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X_df), start=1):
    X_train, X_test = X_df.iloc[train_idx], X_df.iloc[test_idx]
    y_train, y_test = y_df.iloc[train_idx], y_df.iloc[test_idx]

    rf_model.fit(X_train, y_train)
    y_pred = rf_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    accuracies.append(accuracy)
    classification_reports.append(class_report)

    print(f"Fold {fold} Accuracy: {accuracy:.2f}")
    print(f"Fold {fold} Classification Report:")
    print(class_report)

mean_accuracy = np.mean(accuracies)
print(f"Mean Accuracy: {mean_accuracy:.2f}")
```

GENETIC ALGORITHM

Genetic Algorithms (GAs) are algorithms that identify optimum or nearly optimal solutions for optimization and search problems by modelling the concepts of natural selection and evolution. The procedure starts with initialising a population of hypothetical solutions, which are symbolised by chromosomes. The fitness of these chromosomes is assessed by measuring how effectively they meet the goals of the given situation. Similar to the idea of survival of the fittest, during the selection phase, those with better fitness are more likely to be selected as parents for reproduction. In order to produce children, crossover or recombination entails the exchange of genetic information between chosen individuals. Furthermore, mutation adds to genetic variety by causing some individuals to experience random alterations. The original people, crossover children, and modified individuals comprise the new population that replaces the previous population.

Until a termination condition is satisfied, this cycle of crossing, mutation, and selection is continued for a number of generations. Genetic algorithms are very adaptable and broadly applicable. Their performance is contingent upon several factors, including but not limited to population size, crossover rate, mutation rate, and termination criteria. These parameters need meticulous tweaking to achieve peak performance within certain problem domains.

The code implements a genetic algorithm for feature selection in a classification job by using the DEAP library. It initialises a population of feature choices and uses a RandomForestClassifier to assess each one's performance. In order to enhance classification accuracy, the algorithm evolves the population through crossover and mutation over a predetermined number of generations. The best candidate is chosen to represent the chosen qualities, and their indices are produced. For a given issue, we can alter parameters such as the number of features, population size, and generations.

```
import random
import numpy as np
from deap import base, creator, tools, algorithms
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define the number of features (genes) to select
num_features_to_select = X.shape[1]  # You can adjust this number

# Create a fitness function for classification accuracy
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# Initialize the toolbox
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.choice, [True, False])
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=num_features_to_select)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Define the evaluation function (fitness function)
def evaluate(individual):
    selected_features = [feature for feature, select in zip(X_train.T, individual) if select]
    if not any(individual) or len(selected_features) == 0:
        return 0.0,  # If no features are selected, return a fitness of 0
    clf = RandomForestClassifier(random_state=0)
    clf.fit(X_train[:, individual], y_train)
    y_pred = clf.predict(X_test[:, individual])
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy,
```

```python
toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# Create the initial population
population = toolbox.population(n=50)  # You can adjust the population size

# Run the genetic algorithm
generations = 10  # You can adjust the number of generations
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.7, mutpb=0.3)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))

# Find and print the best individual (selected features)
best_individual = tools.selBest(population, k=1)[0]
selected_features_indices = [i for i, select in enumerate(best_individual) if select]
print("Best Individual (Feature Selection):", selected_features_indices)
```

```
Best Individual (Feature Selection): [0, 1, 2, 4, 5, 8]
```

ARTIFICIAL NEURAL NETWORK:

Artificial Neural Networks (ANNs) are computer models made up of linked layers of nodes (neurons) that are inspired by the composition and operations of the human brain. Because ANNs can learn from data and make generalisations, they are useful for a variety of tasks, including categorization. We have built a feedforward neural network for binary classification using TensorFlow's Keras API.

Sequential Model: A linear stack of layers for the neural network is created using the Sequential class. The architecture of the model is defined by the successive addition of layers.

Dense Layers: These are completely linked layers in which every neuron in the layers above and below is connected to every other neuron. With 256 neurons and the ReLU activation function, the first layer in this code (Dense(units=256, activation='relu', input_dim=len(selected_features_indices)) determines the input dimension depending on the chosen features. ReLU activation and a decrease in the number of neurons (128, 64, 64) characterise the subsequent dense layers.

Dropout Layers: To avoid overfitting, dropout layers are sandwiched between dense layers. During training, they arbitrarily remove a portion of the neurons, which forces the network to acquire more resilient features. The designated rates of dropout are 0.2, 0.4, 0.4, and 0.4, in that order.

Output Layer: One neuron with a sigmoid activation function is present in the last layer, Dense(units=1, activation='sigmoid')). This produces probabilities between 0 and 1, making it appropriate for tasks involving binary classification.

Compilation: The learning process is configured using the compile method. During training, the optimizer 'adam' adjusts the learning rate. For tasks involving binary classification, the 'binary_crossentropy' loss function is suitable. 'Accuracy' is a metric that is tracked during training.

Model Summary: The model architecture, including the type and number of layers, the number of parameters, and the output shape at each layer, is succinctly summarised by the function summary().

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

classifier = Sequential()
classifier.add(Dense(units=256, activation='relu', input_dim=len(selected_features_indices)))
classifier.add(Dropout(rate=0.4))
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dropout(rate=0.4))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dropout(rate=0.4))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dropout(rate=0.2))
classifier.add(Dense(units=1, activation='sigmoid'))
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
classifier.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 256)               1792

 dropout_16 (Dropout)        (None, 256)               0

 dense_21 (Dense)            (None, 128)               32896

 dropout_17 (Dropout)        (None, 128)               0

 dense_22 (Dense)            (None, 64)                8256

 dropout_18 (Dropout)        (None, 64)                0

 dense_23 (Dense)            (None, 64)                4160

 dropout_19 (Dropout)        (None, 64)                0

 dense_24 (Dense)            (None, 1)                 65

=================================================================
Total params: 47169 (184.25 KB)
Trainable params: 47169 (184.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

RESULTS

Support Vector Machine (SVM):

Accuracy: 71%

Precision (Class 0/1): 0.73 / 0.71

Recall (Class 0/1): 0.66 / 0.77

F1-Score (Class 0/1): 0.69 / 0.74

Support Vector Machine (SVM) (KFolds):

Fold 1 Accuracy: 73.81%

Fold 2 Accuracy: 68.67%

Fold 3 Accuracy: 67.47%

Fold 4 Accuracy: 75.90%

Fold 5 Accuracy: 61.45%

Fold 6 Accuracy: 71.08%

Fold 7 Accuracy: 80.72%

Mean Accuracy: 71%

Logistic Regression:

Accuracy: 72%

Precision (Class 0/1): 0.68 / 0.74

Recall (Class 0/1): 0.56 / 0.83

F1-Score (Class 0/1): 0.61 / 0.78

Logistic Regression (KFolds):

(Fold 1 Accuracy: 72%

Fold 2 Accuracy: 73%

Fold 3 Accuracy: 70%

Fold 4 Accuracy: 75%

Fold 5 Accuracy: 71%)

Mean Accuracy: 72%

K-Nearest Neighbors (KNN):

Accuracy: 64%

Precision (Class 0/1): 0.48 / 0.72

Recall (Class 0/1): 0.39 / 0.81

F1-Score (Class 0/1): 0.43 / 0.76

K-Nearest Neighbors (KNN) (KFolds):

(Fold 1 Accuracy: 64%

Fold 2 Accuracy: 66%

Fold 3 Accuracy: 63%

Fold 4 Accuracy: 67%

Fold 5 Accuracy: 67%

Fold 6 Accuracy: 59%

Fold 7 Accuracy: 61%)

Mean Accuracy: 64%

Random Forest:

Accuracy: 64%

Precision (Class 0/1): 0.49 / 0.72

Recall (Class 0/1): 0.36 / 0.85

F1-Score (Class 0/1): 0.41 / 0.78

Random Forest (KFolds):

(Fold 1 Accuracy: 71%

Fold 1 Classification Report:

Precision (Class 0/1): 0.38 / 0.83

Recall (Class 0/1): 0.42 / 0.80

F1-Score (Class 0/1): 0.40 / 0.81

Fold 2 Accuracy: 78%

Fold 2 Classification Report:

Precision (Class 0/1): 0.73 / 0.79

Recall (Class 0/1): 0.35 / 0.95

F1-Score (Class 0/1): 0.47 / 0.86

Fold 3 Accuracy: 63%

Fold 3 Classification Report:

Precision (Class 0/1): 0.41 / 0.70

Recall (Class 0/1): 0.33 / 0.77

F1-Score (Class 0/1): 0.37 / 0.74

Fold 4 Accuracy: 64%

Fold 4 Classification Report:

Precision (Class 0/1): 0.33 / 0.74

Recall (Class 0/1): 0.30 / 0.77

F1-Score (Class 0/1): 0.32 / 0.75

Fold 5 Accuracy: 77%

Fold 5 Classification Report:

Precision (Class 0/1): 0.60 / 0.81

Recall (Class 0/1): 0.41 / 0.90

F1-Score (Class 0/1): 0.49 / 0.85

Fold 6 Accuracy: 76%

Fold 6 Classification Report:

Precision (Class 0/1): 0.88 / 0.75

Recall (Class 0/1): 0.27 / 0.98

F1-Score (Class 0/1): 0.41 / 0.85

Fold 7 Accuracy: 65%

Fold 7 Classification Report:

Precision (Class 0/1): 0.43 / 0.70

Recall (Class 0/1): 0.22 / 0.86

F1-Score (Class 0/1): 0.29 / 0.77)

Mean Accuracy: 71%

Artificial Neural Network (ANN):

The training process for the Artificial Neural Network (ANN) is displayed over 50 epochs. The key metrics include loss and accuracy, both for the training set and the validation set. Additionally, a confusion matrix and accuracy score are provided for predictions on a new test set.

Training Process:

The ANN was trained over 50 epochs, and showed a gradual decrease in loss and an increase in accuracy.

The initial loss is 0.6592 with an accuracy of 62.30%, and by the 50th epoch, the loss decreases to 0.5219 with an accuracy of 69.67%.

The validation set follows a similar trend, indicating effective training without significant overfitting.

Confusion Matrix and Accuracy on New Test Set:

The confusion matrix shows that out of 165 instances, 8 true negatives (TN), 116 true positives (TP), 40 false positives (FP), and 11 false negatives (FN) were predicted.

The accuracy on the new test set is calculated as 70.86%.

CONCLUSION

Logistic Regression shows a good balance between accuracy, precision, and recall, making it a strong candidate. Random Forest achieves high precision and recall but with slightly lower accuracy. SVM demonstrates a competitive performance with balanced precision and recall. KNN shows a lower accuracy and F1-Score compared to other algorithms. The choice of the best-performing algorithm depends on the specific goals and priorities.

The ANN exhibits a consistent improvement in both loss reduction and accuracy enhancement over the training epochs.

The confusion matrix suggests a reasonable performance, with a notable number of true positives and a relatively low number of false positives and false negatives.

The overall accuracy on the new test set is 70.86%, indicating a moderately successful prediction capability of the ANN.

Further fine-tuning or exploration of more complex architectures could potentially enhance the model's performance.

KFolds:

SVM: Achieved a consistent accuracy of 71% across multiple folds, with individual fold accuracies ranging from 61.45% to 80.72%.

Logistic Regression: Demonstrated stable accuracy with a mean of 72%.

KNN: Showed variability in accuracy across folds, averaging at 64%.

Random Forest: Random Forest exhibits varying performance across folds, with accuracies ranging from 63% to 78%.

The model demonstrates consistent precision for predicting liver disease (Class 1) but shows variability in recall across folds.

The mean accuracy of 71% suggests stable overall performance across different subsets of the dataset.

Random Forest serves as a reliable model for liver disease prediction, demonstrating competitive performance in KFold cross-validation.

REFERENCES

[1] Nahar, Nazmun & Ara, Ferdous. (2018). Liver Disease Prediction by Using Different Decision Tree Techniques. International Journal of Data Mining & Knowledge Management Process. 8. 01-09. 10.5121/ijdkp.2018.8201.

[2] [Rahman, A. K. M. & Shamrat, F M & Tasnim, Zarrin & Roy, Joy & Hossain, Syed. (2019). A Comparative Study On Liver Disease Prediction Using Supervised Machine Learning Algorithms. 8. 419-422.

[3] V. Singh, M. K. Gourisaria and H. Das, "Performance Analysis of Machine Learning Algorithms for Prediction of Liver Disease," 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), Kuala Lumpur, Malaysia, 2021, pp. 1-7, doi: 10.1109/GUCON50781.2021.9573803.

[4] R. Kalaiselvi, K. Meena and V. Vanitha, "Liver Disease Prediction Using Machine Learning Algorithms," 2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICA ECA), Coimbatore, India, 2021, pp. 1-6, doi: 10.1109/ICAECA52838.2021.9675756.

[5] Kefelegn, S., & Kamat, P. (2018). Prediction and analysis of liver disorder diseases by using data mining technique: survey. International Journal of pure and applied mathematics, 118(9), 765-770.

[6] M. Hassoon, M. S. Kouhi, M. Zomorodi-Moghadam and M. Abdar, "Rule Optimization of Boosted C5.0 Classification Using Genetic Algorithm for Liver disease Prediction," 2017 International Conference on Computer and Applications (ICCA), Doha, Qatar, 2017, pp. 299-305, doi: 10.1109/COMAPP.2017.8079783.

[7] Veeranki, Sreenivasa Rao, and Manish Varshney. "Intelligent Techniques and Comparative Performance Analysis of Liver Disease Prediction." International Journal of Mechanical Engineering 7.1 (2022): 489-503.

[8] Vijayarani, S., and S. Dhayanand. "Liver disease prediction using SVM and Naïve Bayes algorithms." International Journal of Science, Engineering and Technology Research (IJSETR) 4.4 (2015): 816-820.

[9] Devikanniga, D., Arulmurugan Ramu, and Anandakumar Haldorai. "Efficient diagnosis of liver disease using a support vector machine optimized with crows search algorithm." EAI Endorsed Transactions on Energy Web 7.29 (2020): e10-e10.

[10] Priya, M. Banu, P. Laura Juliet, and P. R. Tamilselvi. "Performance analysis of liver disease prediction using machine learning algorithms." Int. Res. J. Eng. Technol 5.1 (2018): 206-211.