2025

Machine Learning Mini Project

FATIN NADIAH MAT ALI (2024474836)

Introduction

This mini project will concentrate on the implementation and comparative analysis of four widely used machine learning algorithms. The selected algorithms are K-Nearest Neighbors (KNN), Naïve Bayes, Support Vector Machine (SVM), and Decision Tree. Each of these algorithms has distinct characteristics and strengths, and the goal of this project is to evaluate their performance in predicting hypertension based on a publicly available dataset.

The dataset utilized in this project is obtained from Figshare, which contains comprehensive physiological and demographic information relevant to hypertension diagnosis. The dataset includes the following features: a numerical identifier (Num.), Subject ID, Sex, Age, Height, Weight, Systolic Blood Pressure (mmHg), Diastolic Blood Pressure (mmHg), Heart Rate, and Body Mass Index (BMI). These attributes collectively offer a well-rounded set of indicators commonly used in clinical assessments of cardiovascular health.

The primary target variable in the dataset is a classification indicating whether an individual has which classification of hypertension or not, based on medical evaluations. This classification serves as the ground truth for training and testing the machine learning models. By leveraging these features, the project aims to develop predictive models that can accurately classify patients into hypertensive or non-hypertensive categories.

In summary, this mini project will explore the practical application of KNN, Naïve Bayes, SVM, and Decision Tree algorithms in the context of hypertension prediction. Through preprocessing, training, evaluation, and comparison of these models, the project will provide insights into their relative effectiveness and potential use in real-world healthcare scenarios.

Methodology

As mentioned in introduction, the algorithms for the model training that will be used is KNN , Naïve Bayes, SVM and Decision Tree. The scripts will be scripted using Mathlab.

KNN - Algorithm that classifies the target based on calculating the distance between the target and other classes. [1]

- **Step 1:** Pick a number **K** (how many neighbours we want to look at).
- **Step 2:** Measure how far the new point is from all other points (using distance).
- **Step 3:** Find the **K** closest points.
- **Step 4:** See which group (or class) most of those **K** points belong to.
- **Step 5:** Put the new point into that group.
- **Step 6:** KNN model is now ready to use.

Naïve Bayes - Naive Bayes is a straightforward and efficient classification algorithm based on Bayes' Theorem. It operates under the assumption that all features used to predict the outcome are independent of one another, even if that may not always be the case in reality. The algorithm calculates the probability of a data point belonging to each possible class and then assigns it to the class with the highest probability.

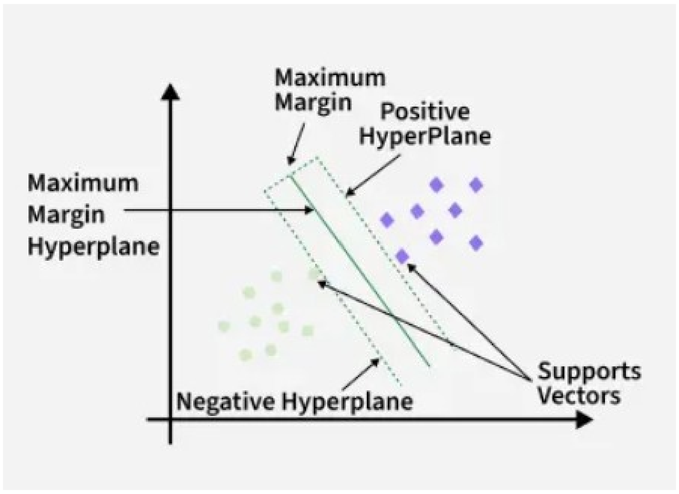
Despite its simplicity, Naive Bayes has proven to be highly effective in many practical applications, particularly in the field of natural language processing (NLP), such as text classification, spam detection, and sentiment analysis. [2]

- **Step 1: Prepare the dataset**
  Start with a labeled dataset where each example includes input features and a target class.
- **Step 2: Convert data into frequency tables (for categorical data)**
  Count how often each feature value appears for each class.
- **Step 3: Calculate prior probabilities**
  Find the overall probability of each class in the dataset:
  $$P(Class) = \frac{\text{Number of samples in class}}{\text{Total number of samples}}$$
- **Step 4: Calculate likelihood (conditional probabilities)**
  For each feature, calculate the probability of a feature value given a class:
  $$P(Feature|Class)$$
- This step assumes all features are independent of each other (the "naive" assumption).
- **Step 5: Apply Bayes' Theorem**
  Use the formula to calculate the posterior probability for each class:
  $$P(Class|Data) = \frac{P(Data|Class) \times P(Class)}{P(Data)}$$
- **Step 6: Choose the class with the highest probability**
  Select the class that has the highest **posterior probability** as the prediction for the new data point.
- **Step 7: Model is ready for prediction**
  You can now use the trained model to classify new, unseen data.

SVM - Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification and regression tasks. Its primary objective is to identify an optimal boundary, known as the hyperplane, that effectively separates different classes within the data.

SVM is particularly effective for binary classification problems, such as distinguishing between spam and non-spam emails or differentiating class A from class B. The algorithm aims to maximize the margin, which is the distance between the hyperplane and the closest data points from each class.

Maximizing this margin helps improve the model's ability to generalize, enabling it to perform well on new, unseen data. [3]



- **Step 1: Prepare the dataset**
  Collect and preprocess the labeled data, making sure the features and target classes are ready for training.
- **Step 2: Choose the kernel function**
  Select a kernel function (e.g., linear, polynomial, radial basis function) that helps the SVM handle data that is not linearly separable.
- **Step 3: Find the optimal hyperplane**
  The SVM algorithm searches for the hyperplane that best separates the classes by maximizing the margin — the distance between the closest points of each class to the hyperplane.
- **Step 4: Identify support vectors**
  Determine the data points closest to the hyperplane, called support vectors, which are crucial in defining the boundary.
- **Step 5: Train the model**
  Use the training data and the selected kernel to fit the SVM model, optimizing the position of the hyperplane.
- **Step 6: Make predictions**
  Apply the trained SVM model to new, unseen data to classify it into the appropriate categories.
- **Step 7: Evaluate the model**
  Assess the model's performance using metrics like accuracy, precision, recall, or F1-score.

Decision Tree - A decision tree is a way for a computer to make decisions by following a flowchart-like structure. It looks like a tree with different parts: a starting point called the root, branches that split off, points where decisions are made (called internal nodes), and endpoints called leaf nodes.

Decision trees help many popular machine learning methods, like Random Forests and Boosting. The tree works by asking questions about the data at each internal node (like yes/no questions), following the branches based on the answers, and finally reaching a leaf node that gives the result or category.

This process is like how tree traversal algorithms work in computer science. Both start at the root and move through the nodes by following branches. While decision trees use this method to make decisions, tree traversal algorithms systematically visit each node in the tree to explore or process the data. [4]

- **Step 1: Prepare the dataset**
  Collect and clean the data, including input features and target labels.
- **Step 2: Choose the best feature to split**
  At each step (node), select the feature that best separates the data into classes. Common measures for this are **Gini impurity**, **Information Gain**, or **Entropy**.
- **Step 3: Split the dataset**
  Divide the dataset into subsets based on the chosen feature's values or conditions.
- **Step 4: Create child nodes**
  For each subset, create a child node and repeat the process of selecting the best feature to split.
- **Step 5: Stop splitting**
  Stop splitting when:
  - All data in a node belong to the same class, or
  - There are no more features to split on, or
  - A pre-set depth limit or minimum number of samples is reached.
- **Step 6: Assign class labels**
  Assign the most common class label in each leaf node.
- **Step 7: Use the tree for prediction**
  To classify new data, start at the root and follow the branches based on the feature values until you reach a leaf node with the predicted class.

Dataset

The dataset comprises comprehensive records that include demographic information (such as subject ID, sex, age, height, and weight), clinical measurements (including systolic and diastolic blood pressure and heart rate), and disease status (notably hypertension and diabetes). Additionally, it contains photoplethysmogram (PPG) waveform data collected under controlled experimental conditions.

To maintain data integrity and reliability, several quality assurance procedures are implemented. These include screening for missing or abnormal physiological and disease-related values, excluding records with incomplete or inconsistent data, and applying a robust signal quality index (SQI) to evaluate and filter out low-quality PPG waveforms.

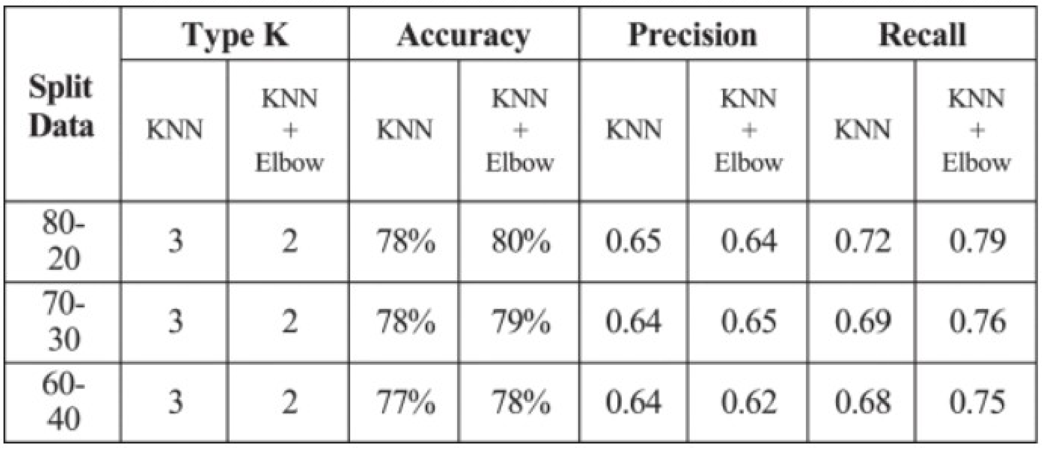
The dataset includes PPG waveforms recorded from subjects under controlled experimental conditions. These waveforms capture real-time blood volume changes, providing essential physiological information related to cardiovascular health. The PPG data serve as the primary input for analyzing and estimating blood pressure and detecting conditions like hypertension. Their non-invasive nature and ease of collection make them particularly valuable for developing machine learning models aimed at early diagnosis and monitoring of cardiovascular diseases.

To ensure the reliability of the PPG waveforms, the dataset employs a robust Signal Quality Index (SQI) as a preprocessing step. The SQI evaluates each PPG signal segment to identify and exclude data contaminated by noise, motion artifacts, or sensor errors. This screening improves the overall quality of the dataset by retaining only high-fidelity signals that accurately reflect physiological conditions. Consequently, the application of SQI enhances the validity and performance of predictive models trained on this dataset, reducing the risk of erroneous conclusions from poor-quality signals. [5]

Experimental Result and Discussion

KNN Configuration

This K configuration is based on the research conducted by Bobby Irawan, Fahmi Fahmi, and Elviawaty Muisa Zamzami, titled *'Optimizing K-Nearest Neighbor Values Using the Elbow Method.* [6] K that has been chosen is 3.



Split Data	Type K		Accuracy		Precision		Recall	
	KNN	KNN + Elbow	KNN	KNN + Elbow	KNN	KNN + Elbow	KNN	KNN + Elbow
80-20	3	2	78%	80%	0.65	0.64	0.72	0.79
70-30	3	2	78%	79%	0.64	0.65	0.69	0.76
60-40	3	2	77%	78%	0.64	0.62	0.68	0.75

Result from the Confusion Matrix are as following :

Actual / Predicted	Normal	Prehypertension	Stage 1	Stage 2
Normal	71 ✓	9 ✗	0	0
Prehypertension	8 ✗	72 ✓	5 ✗	0
Stage 1 hypertension	2 ✗	6 ✗	26 ✓	0
Stage 2 hypertension	0	1 ✗	3 ✗	16 ✓

From the table , the KNN is performing with a reasonable result where for normal, 71 is classified correctly as normal and 9 is incorrectly predicted as pre-hypertension. For Pre-hypertension, 72 is predicted as Pre-hypertension, while 8 incorrectly predicted as Normal and 5 incorrectly predicted as stage 1. While Stage 1, 26 is correctly predicted as Stage 1 hypertension while 2 incorrectly predicted as Normal and 6 is incorrectly predicted asPre-hypertension. For Sgae 2 Hypertension, 16 is correctly predicted and 1 inaccurately predicted as Pre-hypertension while 3 inaccurately predicted as Stage 1.

Metric	Value	Description
Accuracy	85.0%	Overall correct predictions / total samples
Precision	88.0%	Average precision across all classes (treats all classes equally)
Recall	83.0%	Average recall across all classes
F1-Score	85.0%	Average F1-score across all classes
Most Accurate Class	Stage 2 Hypertension (100%)	Best class-level prediction accuracy
Most Confused Classes	Prehypertension & Stage 1 Hypertension	Highest inter-class misclassification

Naïve Bayes

The Naïve Bayes configuration was adjusted by experimenting with different numbers of input features to evaluate its impact on prediction accuracy. This approach is inspired by the research of Nugraha and Widyantoro in their paper titled Student Performance Prediction Using Naive Bayes Algorithm [7].

Result from the confusion matrix after utilizing half of the class :

Actual ↓ / Predicted →	Normal	PreHT	Stage 1 HT	Stage 2 HT	Total
Normal	✓ 10	✗ 13	0	0	23

Actual ↓ / Predicted →	**Normal**	**PreHT**	**Stage 1 HT**	**Stage 2 HT**	**Total**
Prehypertension	✗ 12	✓ 13	0	0	25
Stage 1 Hypertension	✗ 2	✗ 5	0	0	7
Stage 2 Hypertension	✗ 5	✗ 5	0	0	10

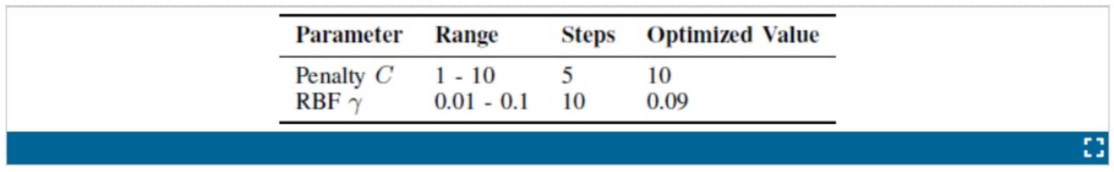
From the table , the Naive Bayes is performing with a poor result where for normal, 10 is classified correctly as normal and 13 is incorrectly predicted as Pre-hypertension. For Pre-hypertension, 13 is predicted as Pre-hypertension, while 12 incorrectly predicted as Normal. None is predicted correctly for stage 1 and stage 2.

Metric	**Value**	**Assessment**
Accuracy	**35.4%**	✗ Poor
Precision	**17.0%**	✗ Needs improvement
Recall	**23.7%**	✗ Needs improvement
F1-Score	**19.8%**	✗ Needs improvement
Most Accurate Class	**Prehypertension (52%)**	⍰ Low but relatively better
Most Confused Classes	**Stage 2 → Normal/PreHT (10)**	✗ Major confusion

Support Vector Machine

Support Vector Machine (SVM) is a classification algorithm that separates data using a hyperplane, which can be linear, parabolic, or based on other kernel functions. The research approach involves introducing a penalty term, known as the slack variable constant **C**, to better tune the model for improved performance. This method is inspired by the study conducted by Abdussalam Mohamed and M. E. El-Hawary, titled *"On Optimization of SVMs Kernels and Parameters for Electricity Price Forecasting."* In this research, the C value is set to 10. [8]

Table II: SVM and kernel parameters optimization



Parameter	Range	Steps	Optimized Value
Penalty C	1 - 10	5	10
RBF γ	0.01 - 0.1	10	0.09

Actual ↓ / Predicted →	**Normal**	**PreHT**	**Stage 1 HT**	**Stage 2 HT**	**Total**
Normal	**23** ✓	0	0	0	23
Prehypertension	1 ✗	**18** ✓	6 ✗	0	25
Stage 1 Hypertension	0	1 ✗	**6** ✓	0	7
Stage 2 Hypertension	0	0	6 ✗	**4** ✓	10

From the table , the SVMis performing with a reasonable result where for normal, 23 is classified correctly as normal. For Pre-hypertension, 18 is predicted as Pre-hypertension, while 1 incorrectly predicted as Normal and 6 incorrectly predicted as stage 1. While Stage 1, 6 is correctly predicted as Stage 1 hypertension while 1 is incorrectly predicted as Pre-hypertension. For Stage 2 Hypertension, 4 is correctly predicted while 6 inaccurately predicted as Stage 1.

Metric	Value	Assessment
Accuracy	**78.5%**	✅ Good
Precision	**80.5%**	✅ Good
Recall	**74.4%**	? Moderate
F1-Score	**70.7%**	? Needs improvement
Most Accurate Class	**Normal (100%)**	✅ Excellent prediction
Most Confused Classes	**Stage 2 → Stage 1** (6 cases)	✕ Major confusion

Decision Tree

Decision Tree is a classification algorithm that makes predictions by traversing its branches and leaves based on input features. One proposed enhancement is to integrate Bayesian decision theory to improve the tree's decision-making process. A notable study that explores this approach is *"A Post-Pruning Decision Tree Algorithm Based on Bayesian,"* conducted by Wenchao Zhang and Yafen Li. Their research applies Bayesian methods for post-pruning to reduce overfitting and improve generalization performance. [9]

Bayesian optimization helps find the best settings for a computer program without trying everything. Instead of guessing randomly, it learns from each try and uses that to make smarter guesses next time. This way, it finds the best answer faster and with less work.

Bayesian optimization remembers all the guesses it has already tried and how good they were. It keeps this information in a special "map" inside the computer's memory. This map helps the computer think about what might work best next.

Actual ↓ / Predicted →	Normal	PreHT	Stage 1 HT	Stage 2 HT	Total
Normal	**23** ✅	0	0	0	23
Prehypertension	0	**25** ✅	0	0	25
Stage 1 Hypertension	0	0	**7** ✅	0	7
Stage 2 Hypertension	0	0	0	**10** ✅	10

For Decision Tree , everything is accurately predicted but there is a suspicion that this model is over fitting due to the unusually perfect prediction.

Metric	Value	Interpretation	Assessment
Accuracy	100% ✅	All 65 predictions were correct	Perfect ✅
Precision	1.000 ✅	Equal average across all classes	Excellent ✅
Recall	1.000 ✅	Perfect average sensitivity	Excellent ✅
F1-Score	1.000 ✅	Perfect balance (P/R) across classes	Excellent ✅

Conclusion

Among the classification models evaluated for hypertension stage prediction, the K-Nearest Neighbors (KNN) classifier demonstrated the best overall performance, achieving an accuracy of 85% with balanced precision, recall, and F1-scores across all classes. In contrast, the Naive Bayes model underperform significantly, with an accuracy of only 35.4% and poor detection of critical stages such as Stage 1 and Stage 2 hypertension. The results suggest that Naive Bayes is not suitable for this task due to high class confusion and limited discriminatory power. Overall, KNN is recommended as the most effective model for this dataset, offering a reliable basis for further development of hypertension classification systems.

Reference

[1] A. Christopher, "K Nearest Neighbor," *Medium*, 2021. [Online]. Available: https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4.

[2] B. Gamal, "Exploring Naïve Bayes: Mathematics, how it works, pros & cons, and applications," *Medium*, 17-Dec-2020. [Online]. Available: https://medium.com/analytics-vidhya/na%C3%AFve-bayes-algorithm-5bf31e9032a2.

[3] GeeksforGeeks, "Support Vector Machine Algorithm," *GeeksforGeeks*, 2023. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/.

[4] A. Bam, "Decision Trees," *Medium*, 2020. [Online]. Available: https://medium.com/@MrBam44/decision-trees-91f61a42c724.

[5] PPG-BP Database, "PPGBP Database," *Figshare*, 2018. [Online]. Available: https://figshare.com/articles/dataset/PPGBP_Database_zip/5459299.

[6] B. Irawan, F. Fahmi, and E. M. Zamzami, "Optimizing K-Nearest Neighbor Values Using The Elbow Method," *in Proceedings of the 2023 5th International Conference on Computer and Communication Engineering (ICCCE)*, IEEE, 2023. doi: 10.1109/ICCCE58086.2023.10957541.

[7] N. Nugraha, D. P. Nugroho, and T. P. Widyantoro, "Student Performance Prediction Using Naive Bayes Algorithm," in *2020 International Conference on Information and Communications Technology (ICOIACT)*, Yogyakarta, Indonesia, 2020, pp. 273–278, doi: 10.1109/ICOIACT48839.2020.9288625.

[8] A. Mohamed and M. E. El-Hawary, "On Optimization of SVMs Kernels and Parameters for Electricity Price Forecasting," in *2016 IEEE Electrical Power and Energy Conference (EPEC)*, Ottawa, Canada, Oct. 2016, pp. 1–6, doi: 10.1109/EPEC.2016.7771773

[9] W. Zhang and Y. Li, "A Post-Pruning Decision Tree Algorithm Based on Bayesian," in Proceedings of 2013 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 2013, pp. XX–YY, doi: 10.1109/[…]6643181.

Appendix (CODE)

KNN Code

```
% Define the file path

filePath = 'C:\Users\User\Downloads\5459299\PPG-BP Database\mini_project\PPG-BP-Dataset.xlsx';

% Create import options

opts = detectImportOptions(filePath, 'DataRange', 'A3:K221'); % Adjust the row range as needed


% Set the variable names manually if needed

opts.VariableNames = {'Num.', 'subject_ID', 'Sex', 'Age', 'Height', ...

                      'Weight', 'Systolic Blood Pressure', 'Diastolic Blood Pressure(', ...

                      'Heart Rate', 'BMI','Hypertension'};

% Read the table using the specified options

T = readtable(filePath, opts);

% Ensure 'Sex' is a categorical variable

T.Sex = categorical(T.Sex);

% Map 'Sex' to numeric values: 1 for male, 2 for female

T.SexNumeric = double(T.Sex == 'Male') + 1;

% Remove the original 'Sex' variable

T = removevars(T, 'Sex');

% Train your model using Hypertension as response

Mdl = fitcknn(T, 'Hypertension', ...

    'NumNeighbors', 3, ...

    'Standardize', true, ...

    'CategoricalPredictors', [], ...

    'Distance', 'euclidean');

% Predict labels on training data

trainPredictions = predict(Mdl, T);

% Calculate accuracy

trainAccuracy = sum(strcmp(trainPredictions, T.Hypertension)) / height(T);

fprintf('Training Accuracy: %.2f%%\n', trainAccuracy * 100);

% Generate confusion matrix

[C, order] = confusionmat(T.Hypertension, trainPredictions);

% Display confusion matrix

disp('Confusion Matrix:');
disp(C);
% Visualize confusion matrix
figure;
cm = confusionchart(C, order);
cm.Title = 'Confusion Matrix for Training Data';
cm.RowSummary = 'row-normalized';
cm.ColumnSummary = 'column-normalized';
% Extract counts from confusion matrix
TP = C(2,2); % True Positives
FP = C(1,2); % False Positives
FN = C(2,1); % False Negatives
TN = C(1,1); % True Negatives

% Calculate metrics
accuracy = (TP + TN) / sum(C(:));
precision = TP / (TP + FP);
recall = TP / (TP + FN);
f1Score = 2 * (precision * recall) / (precision + recall);
% Display metrics
fprintf('Accuracy: %.2f%%\n', accuracy * 100);
fprintf('Precision: %.2f%%\n', precision * 100);
fprintf('Recall: %.2f%%\n', recall * 100);
fprintf('F1-Score: %.2f%%\n', f1Score * 100);
% Display metrics
fprintf('True Positive: %.2f%%\n', TP);
fprintf('True Negative: %.2f%%\n', TN);
fprintf('False Positive: %.2f%%\n', FP);
fprintf('False Negative: %.2f%%\n', FN);
```

SVM Code

```
% ---------------------------------------
% ? Load and preprocess the dataset
filePath = 'C:\Users\User\Downloads\5459299\PPG-BP Database\mini_project\PPG-BP-Dataset.xlsx';
opts = detectImportOptions(filePath, 'DataRange', 'A3:K221');
opts.VariableNames = {'Num_', 'subject_ID', 'Sex', 'Age', 'Height', ...
                      'Weight', 'SystolicBloodPressure', 'DiastolicBloodPressure', ...
                      'HeartRate', 'BMI','Hypertension'};
T = readtable(filePath, opts);
T.Sex = categorical(T.Sex);
```

```
T.SexNumeric = double(T.Sex == 'Male') + 1;

T = removevars(T, 'Sex');

T.Hypertension = categorical(T.Hypertension); % ensure target is categorical

% ---------------------------------------

% ? Split: 70% training, 30% testing

rng(1);

cv = cvpartition(height(T), 'HoldOut', 0.3);

trainData = T(training(cv), :);

testData = T(test(cv), :);

% Define the SVM template with linear kernel and box constraint (e.g., C = 1)

t = templateSVM('KernelFunction', 'linear', 'BoxConstraint', 10);

% ---------------------------------------

% ? Train SVM Classifier

SVMModel = fitcecoc(trainData, 'Hypertension', ...

               'Learners', t, ...

               'ClassNames', categories(T.Hypertension), ...

               'Coding', 'onevsall'); % Recommended for multi-class or categorical response

% Predict on test data

testPredictions = predict(SVMModel, testData);

% ---------------------------------------

% ? Evaluation Metrics

accuracy = sum(categorical(testPredictions) == categorical(testData.Hypertension)) / numel(testPredictions);

fprintf('\n? SVM Performance Metrics:\n');

fprintf('? Test Accuracy: %.2f%%\n', accuracy * 100);

% Confusion matrix

classLabels = categories(categorical(T.Hypertension));

confMat = confusionmat(categorical(testData.Hypertension), categorical(testPredictions), ...

                'Order', categorical(classLabels));

disp('? Confusion Matrix:');

disp(array2table(confMat, ...

    'VariableNames', strcat('Pred_', classLabels), ...

    'RowNames', strcat('Actual_', classLabels)));

% Optional: binary metrics

if numel(classLabels) == 2

    TP = confMat(2,2);

    TN = confMat(1,1);

    FP = confMat(1,2);

    FN = confMat(2,1);

    precision = TP / (TP + FP);

    recall = TP / (TP + FN);
    f1score = 2 * (precision * recall) / (precision + recall);
    fprintf('? Precision: %.2f%%\n', precision * 100);
    fprintf('? Recall: %.2f%%\n', recall * 100);
    fprintf('? F1-score: %.2f%%\n', f1score * 100);
else
    fprintf('?? Precision/Recall/F1 only shown for binary classification.\n');
end
% ---------------------------------------
% ?? Plot Decision Surface (Age vs BMI)
patientAge = trainData.Age;
patientBMI = trainData.BMI;
x1range = min(patientAge):0.5:max(patientAge);
x2range = min(patientBMI):0.5:max(patientBMI);
[xx1, xx2] = meshgrid(x1range, x2range);
XGrid = [xx1(:), xx2(:)];
% Prepare dummy table with fixed average values
TblGrid = table(...
    repmat(mean(trainData.Num_), size(XGrid,1), 1), ...
    repmat(mean(trainData.subject_ID), size(XGrid,1), 1), ...
    XGrid(:,1), XGrid(:,2), ...
    repmat(mean(trainData.Height), size(XGrid,1),1), ...
    repmat(mean(trainData.Weight), size(XGrid,1),1), ...
    repmat(mean(trainData.SystolicBloodPressure), size(XGrid,1),1), ...
    repmat(mean(trainData.DiastolicBloodPressure), size(XGrid,1),1), ...
    repmat(mean(trainData.HeartRate), size(XGrid,1),1), ...
    repmat(mean(trainData.SexNumeric), size(XGrid,1),1), ...
    'VariableNames', {'Num_','subject_ID','Age','BMI','Height','Weight', ...
                    'SystolicBloodPressure','DiastolicBloodPressure','HeartRate','SexNumeric'});
% Predict over grid
predGrid = predict(SVMModel, TblGrid);
predGridNum = double(categorical(predGrid));
figure;
contourf(xx1, xx2, reshape(predGridNum, size(xx1)), 'LineStyle', 'none');
colormap([0.8 0.92 1; 1 0.8 0.8]);
hold on;
% Plot training data points
if ~isempty(trainData.Age) && ~isempty(trainData.BMI) && ~isempty(trainData.Hypertension)
    hypertensionCat = categorical(trainData.Hypertension);
    hData = gscatter(trainData.Age, trainData.BMI, hypertensionCat, 'rb', 'xo');
else
```

```
    warning('Train data for Age, BMI, or Hypertension is empty or invalid.');
end
% Create dummy patch handles for decision surface legend
p1 = patch(nan, nan, [0.8 0.92 1]);
p2 = patch(nan, nan, [1 0.8 0.8]);
% Compose legend
legendHandles = [p1, p2, hData];
legendLabels = {'No Hypertension Region', 'Hypertension Region', ...
                'No Hyper Data', 'Hyper Data'};

legend(legendHandles, legendLabels, 'Location', 'Best');
xlabel('Age');
ylabel('BMI');
title('SVM Decision Surface (Age vs BMI)');
hold off;
```

Decision Tree Code

```
% ---------------------------------------
% ? Load and preprocess the dataset
filePath = 'C:\Users\User\Downloads\5459299\PPG-BP Database\mini_project\PPG-BP-Dataset.xlsx';
opts = detectImportOptions(filePath, 'DataRange', 'A3:K221');
opts.VariableNames = {'Num_', 'subject_ID', 'Sex', 'Age', 'Height', ...
                      'Weight', 'SystolicBloodPressure', 'DiastolicBloodPressure', ...
                      'HeartRate', 'BMI','Hypertension'};
T = readtable(filePath, opts);
T.Sex = categorical(T.Sex);
T.SexNumeric = double(T.Sex == 'Male') + 1;
T = removevars(T, 'Sex');
T.Hypertension = categorical(T.Hypertension); % ensure target is categorical
% ---------------------------------------
% ? Split: 70% training, 30% testing
rng(1);
cv = cvpartition(height(T), 'HoldOut', 0.3);
trainData = T(training(cv), :);
testData = T(test(cv), :);
opts = struct(...
    'Optimizer', 'bayesopt', ...
    'ShowPlots', true, ...
    'UseParallel', true, ...
    'MaxObjectiveEvaluations', 30, ...
    'Verbose', 1);

treeModel = fitctree(trainData, 'Hypertension', ...
    'CategoricalPredictors', {'SexNumeric'}, ...
    'ClassNames', categories(T.Hypertension), ...
    'OptimizeHyperparameters', {'MaxNumSplits', 'MinLeafSize', 'SplitCriterion'}, ...
    'HyperparameterOptimizationOptions', opts);
% Predict on test data
testPredictions = predict(treeModel, testData);
% ---------------------------------------
% ? Evaluation Metrics
accuracy = sum(categorical(testPredictions) == categorical(testData.Hypertension)) / numel(testPredictions);
fprintf('\n? Decision Tree Performance Metrics:\n');
fprintf('? Test Accuracy: %.2f%%\n', accuracy * 100);
% Confusion matrix
classLabels = categories(categorical(T.Hypertension));
confMat = confusionmat(categorical(testData.Hypertension), categorical(testPredictions), ...
                    'Order', categorical(classLabels));
disp('? Confusion Matrix:');
disp(array2table(confMat, ...
    'VariableNames', strcat('Pred_', classLabels), ...
    'RowNames', strcat('Actual_', classLabels)));
% Optional: binary metrics
if numel(classLabels) == 2
    TP = confMat(2,2);
    TN = confMat(1,1);
    FP = confMat(1,2);
    FN = confMat(2,1);
    precision = TP / (TP + FP);
    recall = TP / (TP + FN);
    f1score = 2 * (precision * recall) / (precision + recall);
    fprintf('? Precision: %.2f%%\n', precision * 100);
    fprintf('? Recall: %.2f%%\n', recall * 100);
    fprintf('? F1-score: %.2f%%\n', f1score * 100);
else
    fprintf('?? Precision/Recall/F1 only shown for binary classification.\n');
end
% ---------------------------------------
% ?? Plot Decision Surface (Age vs BMI)
patientAge = trainData.Age;
```

```
patientBMI = trainData.BMI;
x1range = min(patientAge):0.5:max(patientAge);
x2range = min(patientBMI):0.5:max(patientBMI);
[xx1, xx2] = meshgrid(x1range, x2range);
XGrid = [xx1(:), xx2(:)];
% Prepare dummy table with fixed average values
TblGrid = table(...
    repmat(mean(trainData.Num_), size(XGrid,1), 1), ...
    repmat(mean(trainData.subject_ID), size(XGrid,1), 1), ...
    XGrid(:,1), XGrid(:,2), ...
    repmat(mean(trainData.Height), size(XGrid,1),1), ...
    repmat(mean(trainData.Weight), size(XGrid,1),1), ...
    repmat(mean(trainData.SystolicBloodPressure), size(XGrid,1),1), ...
    repmat(mean(trainData.DiastolicBloodPressure), size(XGrid,1),1), ...
    repmat(mean(trainData.HeartRate), size(XGrid,1),1), ...
    repmat(mean(trainData.SexNumeric), size(XGrid,1),1), ...
    'VariableNames', {'Num_','subject_ID','Age','BMI','Height','Weight', ...
                  'SystolicBloodPressure','DiastolicBloodPressure','HeartRate','SexNumeric'});
% Predict over grid
predGrid = predict(treeModel, TblGrid);
predGridNum = double(categorical(predGrid));
figure;
contourf(xx1, xx2, reshape(predGridNum, size(xx1)), 'LineStyle', 'none');
colormap([0.8 0.92 1; 1 0.8 0.8]);
hold on;
% Plot training data points
if ~isempty(trainData.Age) && ~isempty(trainData.BMI) && ~isempty(trainData.Hypertension)
    hypertensionCat = categorical(trainData.Hypertension);
    hData = gscatter(trainData.Age, trainData.BMI, hypertensionCat, 'rb', 'xo');
else
    warning('Train data for Age, BMI, or Hypertension is empty or invalid.');
end
% Create dummy patch handles for decision surface legend
p1 = patch(nan, nan, [0.8 0.92 1]);
p2 = patch(nan, nan, [1 0.8 0.8]);
% Compose legend
legendHandles = [p1, p2, hData];
legendLabels = {'No Hypertension Region', 'Hypertension Region', ...
            'No Hyper Data', 'Hyper Data'};
legend(legendHandles, legendLabels, 'Location', 'Best');
```

```
xlabel('Age');
ylabel('BMI');
title('Decision Tree Decision Surface (Age vs BMI)');
hold off;
```

Naive Bayes Code

```
% --------------------------------------
% ? Load and preprocess the dataset
filePath = 'C:\Users\User\Downloads\5459299\PPG-BP Database\mini_project\PPG-BP-Dataset.xlsx';
opts = detectImportOptions(filePath, 'DataRange', 'A3:K221');
opts.VariableNames = {'Num_', 'subject_ID', 'Sex', 'Age', 'Height', ...
                      'Weight', 'SystolicBloodPressure', 'DiastolicBloodPressure', ...
                      'HeartRate', 'BMI','Hypertension'};
T = readtable(filePath, opts);
T.Sex = categorical(T.Sex);
T.SexNumeric = double(T.Sex == 'Male') + 1;
T = removevars(T, 'Sex');
T.Hypertension = categorical(T.Hypertension); % ensure target is categorical
% --------------------------------------
% ? Split: 70% training, 30% testing
rng(1);
cv = cvpartition(height(T), 'HoldOut', 0.3);
trainData = T(training(cv), :);
testData = T(test(cv), :);
% --------------------------------------
% ?? Select half the features for training
selectedFeatures = {'Age', 'BMI', 'SexNumeric', 'HeartRate', 'Weight'};
% Create reduced training and testing tables
predictorTableTrain = trainData(:, selectedFeatures);
predictorTableTrain.Hypertension = trainData.Hypertension;
predictorTableTest = testData(:, selectedFeatures);
predictorTableTest.Hypertension = testData.Hypertension;
% --------------------------------------
% ? Train Naive Bayes Classifier
categoricalVars = {'SexNumeric'};
nbModel = fitcnb(predictorTableTrain, 'Hypertension', ...
   'CategoricalPredictors', categoricalVars, ...
```

```
    'DistributionNames', 'mvmn', ...
    'ClassNames', categories(T.Hypertension));
% Predict on test data
testPredictions = predict(nbModel, predictorTableTest);
% ---------------------------------------
% ? Evaluation Metrics
accuracy = sum(categorical(testPredictions) == categorical(predictorTableTest.Hypertension)) / numel(testPredictions);
fprintf('\n? Naive Bayes Performance Metrics:\n');
fprintf('? Test Accuracy: %.2f%%\n', accuracy * 100);
% Confusion matrix
classLabels = categories(categorical(T.Hypertension));
confMat = confusionmat(categorical(predictorTableTest.Hypertension), categorical(testPredictions), ...
                       'Order', categorical(classLabels));
disp('? Confusion Matrix:');
disp(array2table(confMat, ...
    'VariableNames', strcat('Pred_', classLabels), ...
    'RowNames', strcat('Actual_', classLabels)));
% Optional: binary metrics
if numel(classLabels) == 2
    TP = confMat(2,2);
    TN = confMat(1,1);
    FP = confMat(1,2);
    FN = confMat(2,1);
    precision = TP / (TP + FP);
    recall = TP / (TP + FN);
    f1score = 2 * (precision * recall) / (precision + recall);
    fprintf('? Precision: %.2f%%\n', precision * 100);
    fprintf('? Recall: %.2f%%\n', recall * 100);
    fprintf('? F1-score: %.2f%%\n', f1score * 100);
else
    fprintf('?? Precision/Recall/F1 only shown for binary classification.\n');
end
% ---------------------------------------
% ?? Plot Decision Surface (Age vs BMI)
patientAge = predictorTableTrain.Age;
patientBMI = predictorTableTrain.BMI;
x1range = min(patientAge):0.5:max(patientAge);
```

```
x2range = min(patientBMI):0.5:max(patientBMI);

[xx1, xx2] = meshgrid(x1range, x2range);

XGrid = [xx1(:), xx2(:)];

% Dummy inputs for other selected features

TblGrid = table(...

    XGrid(:,1), XGrid(:,2), ...

    repmat(mean(predictorTableTrain.SexNumeric), size(XGrid,1),1), ...

    repmat(mean(predictorTableTrain.HeartRate), size(XGrid,1),1), ...

    repmat(mean(predictorTableTrain.Weight), size(XGrid,1),1), ...

    'VariableNames', {'Age','BMI','SexNumeric','HeartRate','Weight'});

% Predict over grid

predGrid = predict(nbModel, TblGrid);

predGridNum = double(categorical(predGrid));

figure;

contourf(xx1, xx2, reshape(predGridNum, size(xx1)), 'LineStyle', 'none');

colormap([0.8 0.92 1; 1 0.8 0.8]);

hold on;

% Plot training data points

if ~isempty(patientAge) && ~isempty(patientBMI) && ~isempty(predictorTableTrain.Hypertension)

    hypertensionCat = categorical(predictorTableTrain.Hypertension);

    hData = gscatter(patientAge, patientBMI, hypertensionCat, 'rb', 'xo');

else

    warning('Train data for Age, BMI, or Hypertension is empty or invalid.');

end

% Create dummy patch handles for decision surface legend

p1 = patch(nan, nan, [0.8 0.92 1]);

p2 = patch(nan, nan, [1 0.8 0.8]);

% Compose legend

legendHandles = [p1, p2, hData];

legendLabels = {'No Hypertension Region', 'Hypertension Region', ...

                'No Hyper Data', 'Hyper Data'};

legend(legendHandles, legendLabels, 'Location', 'Best');

xlabel('Age');

ylabel('BMI');

title('Naive Bayes Decision Surface (Age vs BMI)');

hold off;
```