2022

Intelligent Systems (KF5042) Submission

WORD COUNT: (1120) PART A, () PART B

AXEL.MATTHEW.LIM | ID: W20015790 | 04/04/2022

Repository for project located at: https://github.com/AxelMtt/Intelligent-Systems

Machine Learning Technologies in the Diagnosis of Heart Disease Risks

Using artificial intelligence to simulate approximate risk percentiles of heart diseases caused by personal factors

Axel Matthew Lim
Computer and Information Sciences
Northumbria University at Newcastle
Newcastle upon Tyne, United Kingdom
axel.lim@northumbria.ac.uk

***Abstract*—Recognizing the critical issue of heart disease and complications in the modern era, an exploration of methods regarding its prediction and diagnosis is viewed as vital. Modern science employs machine learning and artificial intelligence technologies as one of the methods to tackle this problem. As a growingly popular field of study, many different experiments and theory papers have been performed and written on the subject, often with differing methodologies or applications. This paper seeks to once again experiment with similar technologies and confirm their operation and theoretical application to predicting heart diseases, keeping in mind the theme of personal habits that affect the odds of heart disease risks. Using an appropriate dataset, the given features are treated to fit the experiment's goal more while potentially sacrificing some degree of accuracy.**

Through a few algorithms hypothesized to give the best results, a model is trained using a subset of data for training, and tested against a different smaller subset, this results in the possibility to evaluate the different models including decision tree, logistic regression, support vector machines, and boosted trees. After considering both speed and accuracy, while also keeping in mind interpretability of the training process and result. The best method for this particular case is found to be both logistic regression and boosted decision trees, with the former providing a better degree of interpretability for advanced users.

***Keywords*—heart disease, artificial intelligence, training data, models, decision tree, logistic regression, support vector machines, boosted trees.**

I. Introduction

Heart diseases, or CVD (Cardiovascular Diseases) has persisted across the years as one of the major leading causes of deaths worldwide in developed nations. In the UK alone, this umbrella term covering all types of heart and circulatory diseases account for 24% of all deaths [1]. Disregarding the recent COVID-19 outbreak, it is arguably one of the most common and deadly health risk experienced by demographics of various ages.

These risks are further exacerbated by the growing popularity of unhealthy lifestyles among young people. Habits and pleasures relating to the consumption of "dirty" fat, lack of exercise, alcoholism, and smoking contributes as major common habits that risks CVD [2]. All of which, has not yet taken into consideration other factors such as those racial or hereditary.

It is due to the condition's severity that the ability to infer at-risk patients with significant accuracy in a short span of time becomes so important. To achieve this goal, numerous prior research papers and experiments delved into the topic of machine learning and AI technologies. Many of them explored both new and similar approaches with different methods of processing features and modelling, while some use completely different attributes from one another. Popular models and approaches included naïve bayes, logistic regressions, and support vector machines.

The medical domain primarily makes use of an AI system's capability of guided prediction (making informed decisions deducting future outcomes based on past data) on a large scale. Early detection/assessment of such health conditions would contribute much to the likelihood of preventative treatments and procedures, while also providing proper palliative care to those in much less fortunate circumstances.

Within the author's limited knowledge, this paper will attempt to review and re-experiment on the same field using similar machine learning technologies, documenting the process done to prepare the dataset and perform trials to deduce a functional model using MATLAB software. Features focused on within these experiments will primarily relate to personal habits such as exercise and refrain from delving into many technical characteristics that require deep expert-level understanding.

II. Literature review

Different approaches of implementing AI technologies have been used to tackle this ongoing field. Azra'ai et al. provides an example of a highly technical implementation of such research, applying the potential of a neural network to evaluate heart risks through the use of "heart sounds" as its feature [3]. A large part of harnessing ML in said experiment

seems to be spent on some degree of visual recognition. Results from testing were fed into a model trained to perform classification heart sound patterns into eleven categories. However, while this implementation is highly interesting, this paper will not attempt to perform the same level of technical experimentation.

In contrast, another research used simpler or more general features as their basis [4], opting to model a decision-making system through machine learning and data mining to locate important identifiers. This data is then treated using principal component analysis to further prune them in an attempt of reducing features/dimensional reduction as this domain prioritizes some degree of speed. A conventional random forest of decision trees is then applied to the clustered form of said aggregate, testing different orders of criteria to select the best possible fitting decision path.

While many other research chose to classify the results of their models into a binary yes/no output, Yuan et al. attempted to instead group these into multiple classes based on angiographic testing results from level 0 (normal) to 4 [5]. The researchers further used gradient boosted trees to classify the processed data, with the basis of it being one of the most proven approach for the situation as it provides an extensive test through variations of different decision trees.

One of the most popular dataset used for experiments in this field is the heart disease dataset donated to the UCI machine learning database [6]. However, this experiment will not source its data from said repository, as it requires more knowledge of the features and belongs to a different scope. Instead, this paper would like to focus on the more personal factor of heart disease risk factors such as those relating to habits, and so will seek to use a more appropriate dataset with non-technical features.

There are several AI models/algorithms that will be considered when picking a proper method for this experiment. The decision tree model would prove useful for feature data types that are less real-ranged and exists more categorically as set values as they provide more consistent decision paths. It essentially works as a set of conditional statements that calculate the values of each feature and summarizes the result as a classifier.

Logistic regression and support vector machines both work well on data sets with well-bounded classifiers. They will provide a clear division between the different classes resulting from the model's prediction. In this case, it works particularly fittingly with the classifier of binary nature (yes/no to heart disease risks).

Neural networks are well known to work as black boxes that excel in learning correlations/patterns between provided data, mimicking a process similar to human thought. It may function well in classifying the risk features introduced, though using it to conduct further study on the features themselves may be difficult as neural networks are harder to interpret.

III. PLANNED EXPERIMENTS

A. Processing the dataset

The experiment will be prefaced by a plan for treatment of the dataset used. The dataset source in question [7] contains several features. While most if not all of the features/attributes present in the dataset is relevant for the training topic of heart disease predictions, this experiment would like to focus on the factors of personal habits. As risk factors among individuals for heart diseases may vary [8], the author has decided to prune certain features and exclude them from the training model altogether (also to focus on the theme of personal habits, which do not rely on a lack of variance between patients). These are chosen on the basis of not fitting the experiment's theme, or redundancy that may be represented by similar features.

The dataset's treatment also includes a consideration of the classifier imbalance problem. The resulting class derived from its features are binary values of "yes" or "no" which refers to the existence or lack of a heart disease risk. The dataset provided has a massive ratio of those without heart disease compared to those with. This could create an inaccuracy in the resulting model due to a bias taking up the majority of the population. To remedy this to a degree, the data could be randomly oversampled (creating relevant dummy data of the class with less instances) or undersampled (pick entries of the overpopulated class to remove at random). The latter option is chosen as it bears less risk of creating problematic new data entries without the proper technical knowledge and provides a faster training time to create the initial models.

The two above processes will be done semi-manually through Microsoft Excel to ensure proper processing. Entries with the "No" value classifier is randomly removed to match the approximate amount of "Yes" classifiers. This ensures a

The features removed include the Sex, Race, KidneyDisease, and SkinCancer feature columns as they are not considered to be as relevant as factors for personal habits, and both medical conditions could be represented by the Diabetes feature for the purpose of "possible medical condition affecting risks" in the dataset.

One last step to prepare the data is splitting it between training data and data for testing purposes to be applied on the trained model later. As a consequence of these treatments, the accuracy or precision of the model's prediction capability will likely see a decline. This is due to having less instances of previous data for the model to learn from, and less features to base the outcome's classifier.

B. Importing the data

The treated workbook is fed into MATLAB using its built-in import tool. A function is generated automatically to repeat this same process importing the test data. Both function calls are put in the experiment script for the sake of automation. The resulting variables should generate a 38797x14 table for the training data, and a 15999x14 table for the test data. The features and classifier are provided in detail in Fig. 1.

C. Training the model

The training process makes use of MATLAB's Classification Learner application to automatically process the input data through different model types. After thorough consideration, the training will only include the decision tree, logistic regression, support vector machines, and boosted decision trees model types. These models are expected to produce an algorithm that predicts the appropriate classifiers inferred from provided features. They will be evaluated based on their time and accuracy performance.

A decision tree and boosted decision trees is suitable for this case, as many of the available features involve a degree of decision making due to a lack of real number ranges. Logistic regressions and support vector machines are both well-established model types for this field and excel in processing datasets that are expected to have clear differences between classes (in this case, "yes" and "no").

BMI: Body Mass Index of the individual

Smoking: Whether the individual is a habitual smoker

AlcoholDrinking: Whether the individual routinely consumes alcoholic beverages

Stroke: Whether the individual has a history of stroke episodes

PhysicalHealth: How many out of the last 30 days has the individual experienced physical problems

MentalHealth: How many out of the last 30 days has the individual experienced mental problems

DiffWalking: If the individual has difficulty walking

AgeCategory: General age range of the individual

Diabetic: Whether the individual has a diabetic disposition

PhysicalActivity: Whether the individual routinely participates in physical activity

GenHealth: General view on personal health of the individual

SleepTime: Approximate time for rest allocated by each individual

Asthma: Whether the individual has a record of asthma conditions

Fig. 1. Features used

This experiment foregoes the use of naïve bayes methods as it is unsuitable in a data environment where certain features may logically have different weighing than others. The algorithm assumes that every feature is completely independent with the same importance, unlike a model such as logistic regression which is possibly used to recognize feature importance. Importance of features should ideally be taken into consideration, or at least interpretable as this field of study could very well make use of the knowledge involved derived from the dataset.

The use of neural networks is also forgone as it is resource intensive and involves a mostly black-box process, preventing proper interpretation by human actors for use outside of the experiment or system.

D. Hypothesis

While decision trees and logistic regression models would likely provide the fastest training times, SVMs and boosted decision trees are highly likely to result in a much more accurate prediction.

Certain visualisation diagrams of relationships between features may look significantly segregated, especially those with "yes" and "no" or binary nature, while some others may possibly present a more spread out/less linearly bounded distribution.

IV. Results analysis

A. Speed and Accuracy

As the results shown in TABLE I., the standard decision tree model (specifically a fine tree method) executed with MATLAB provides by far the shortest time to process a model. In exchange for a shorter training time, it also boasts a consistently lower accuracy (albeit slightly) than the other three models.

An outlier in the completely different direction is the Support Vector Machine model. Compared to other algorithms, it takes a massive 80.21 second average to train, though of course using the existing trained model for actual application would take less time (as there is likely no more, or less training involved).

Between the two remaining models, the average accuracy does not show a significant difference in performance. While the logistic model does possess a more than twice faster speed than boosted tree, it is also worth considering their interpretability as decision nodes in boosted tree models may become overwhelming to observe as their count increases, while feature importance is easily recognizable in logistic models (though feature importance is not something studied in this experiment).

With these points in mind, further discussion will be focused on the logistic regression model, as it proves to be the most well-rounded algorithm through this experiment.

TABLE I. MODEL PERFORMANCE (5 RUNS AVERAGED)

Parameter	Model Type

	Decision Tree	*Logistic*	*SVM*	*Boosted Tree*
Accuracy (Validation)	74.7%	75.38%	75.37%	**75.41%**
Accuracy (Test)	74.8%	**75.61%**	75.57%	75.59%
Time	**0.34 seconds**	1.49 seconds	80.21 seconds	3.51 seconds
Testing Time	**0.21 Seconds**	0.27 seconds	1.93 seconds	0.28 seconds

Disclaimer: Results are very likely going to vary between machines depending on resources and processing power

B. Figure Analysis

Figures were harvested through an automatic toolbox during model training (MATLAB's Classifier Leaner). Due to the nature of most parameters/features having set values instead of a real number range, some diagrams are hardly usable for interpretation. Further discussion will only be done on few select figures that produces a meaningful interpretation.

Figure 2 shows the relationship of some features in the logistic regression model. The first of which shows a clear relationship where most negative heart disease risks identified (true or otherwise) is concentrated on those with minimal physical or mental burden in the past 30 days.

Correct predictions on Fig. 3 are also interpretable to identify the relationships between BMI, physical health, and heart disease risks. While both true positives and false negatives are significantly distributed on the lower half of BMIs for most, true positives of heart disease risk shows a much higher occurrences of higher BMI and a spread of physical health problem reports compared to the lower risks that are focused on lower BMIs with less days of physical health problems.

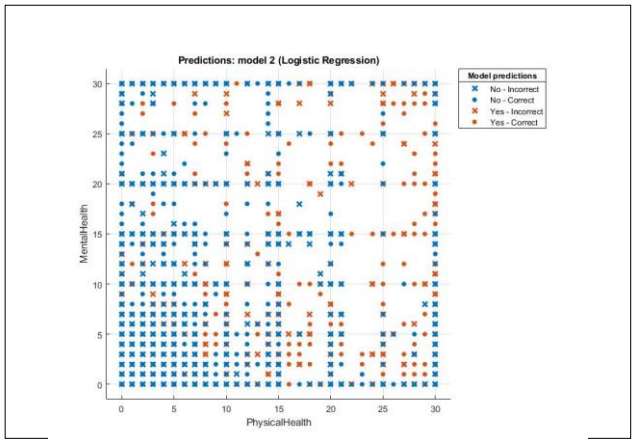

Fig. 2. MentalHealth-PhysicalHealth features relationship

V. Conclusion and Future Work

A. Conclusion

Through the experiment's result, it is concluded that for this particular approach of detecting health disease risks, both the logistic regression and boosted tree model wields the best performance. It is then up to those with the proper expertise and technical knowledge of the field to decide which model is more suitable for interpreting the specific features used in this research.

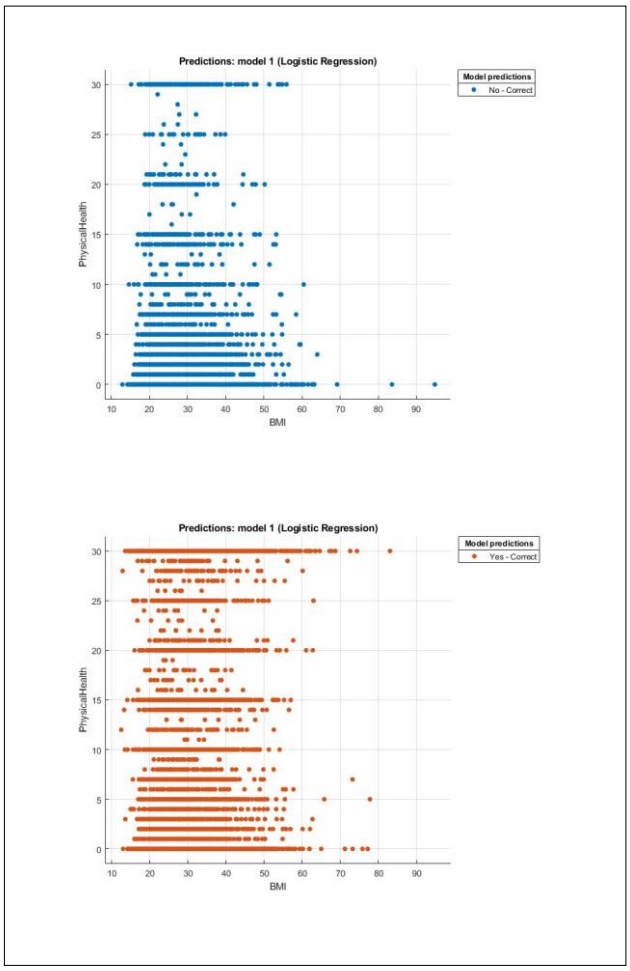

Fig. 3. True negatives compared to true positives on the relationship of BMI and PhysHealth features

The process also results in the identification of several significant factors that increases heart disease risks, namely the relations between body mass index and personal health.

B. Future Work

As mentioned in the conclusion, further study with proper technical expertise is needed to decide which model would suit interpretability better for this particular set of attributes/parameters. With the growing practice of artificial intelligence, it is also possible to discover further untested models and algorithms to tackle this particular set of data, perhaps improving on either/both of its speed and accuracy.

References

[1] BHF, "Heart Statistics", *British Heart Foundation*, 2022. [Online]. Available: https://www.bhf.org.uk/what-we-do/our-research/heart-statistics?gclsrc=aw.ds. [Accessed: 20- Apr- 2022].

[2] CDC, "Know Your Risk for Heart Disease", *Centers for Disease Control and Prevention*, 2019. [Online]. Available: https://www.cdc.gov/heartdisease/risk_factors.htm. [Accessed: 20- Apr- 2022].

[3] R. Azra'ai, M. Taib and N. Tahir, "Artificial Neural Network for identification of heart problem", *2008 2nd International Conference on Signal Processing and Communication Systems*, 2008. Available: 10.1109/icspcs.2008.4813706 [Accessed 20 April 2022].

[4] A. Pati, M. Parhi and B. Pattanayak, "IDMS: An Integrated Decision Making System for Heart Disease Prediction", *2021 1st Odisha International Conference on Electrical Power Engineering, Communication and Computing Technology (ODICON)*, 2021. Available: 10.1109/odicon50556.2021.9428958 [Accessed 20 April 2022].

[5] X. Yuan, J. Chen, K. Zhang, Y. Wu and T. Yang, "A Stable AI-Based Binary and Multiple Class Heart Disease Prediction Model for IoMT", *IEEE Transactions on Industrial Informatics*, vol. 18, no. 3, pp. 2032-2040, 2022. Available: 10.1109/tii.2021.3098306 [Accessed 20 April 2022].

[6] A. Janosi, W. Steinbrunn, M. Pfisterer and R. Detrano, "UCI Machine Learning Repository: Heart Disease Data Set", Archive.ics.uci.edu, 1988. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/heart+disease. [Accessed: 21- Apr- 2022].

[7] K. Pytlak, "Personal Key Indicators of Heart Disease", Kaggle.com, 2020. [Online]. Available: https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease. [Accessed: 02- May- 2022].

[8] P. Schnohr, "Coronary heart disease risk factors ranked by importance for the individual and community. A 21 year follow-up of 12000 men and women from The Copenhagen City Heart Study", *European Heart Journal*, vol. 23, no. 8, pp. 620-626, 2002. Available: 10.1053/euhj.2001.2842 [Accessed 20 April 2022].

ADDENDUM

A. ADDENDUM I: EXPERIMENT EXECUTION SCRIPT

```
fprintf("Intelligent Systems Project by
Axel Lim 20015790\n");

fprintf("Importing Datasets... \n");

%%Import the training worksheet from the
dataset
trainingSet =
ass_import("ass_processed","TrainingSet")
;

%%Import the testing worksheet from the
dataset
testSet =
ass_import("ass_processed","TestSet",[2,
16000]);

fprintf("Import Done! \n");


fprintf("Training Models... \n");

%% train the model and time it
tic
[trainedTree, validationAcc_tree] =
ass_traintree(trainingSet);
timed = toc;
fprintf("Tree Training Done: %f
seconds\n", timed);
tic
[trainedLog, validationAcc_log] =
ass_trainlog(trainingSet);
timed = toc;
fprintf("Logistic Training Done: %f
seconds\n", timed);
tic
[trainedsvm, validationAcc_svm] =
ass_trainsvm(trainingSet);
timed = toc;
fprintf("SVM Training Done: %f
seconds\n", timed);
tic
[trainedBoosted, validationAcc_boosted] =
ass_trainboosted(trainingSet);
timed = toc;
fprintf("Boosted Tree Training Done: %f
seconds\n", timed);

%% display the training validation
accuracy
fprintf("Validation Accuracy for Decision
Tree: %2f \n", validationAcc_tree);
fprintf("Validation Accuracy for Logistic
Regression: %2f \n", validationAcc_log);
fprintf("Validation Accuracy for Support
Vector Machines: %2f \n",
validationAcc_svm);
fprintf("Validation Accuracy for Boosted
Decision Trees: %2f \n",
validationAcc_boosted);

fprintf("Training Done! \n");

%% test data
fprintf("Testing... \n");

source = testSet(:,"HeartDisease");
tic
fprintf("Accuracy on test data for Tree =
%f , done in %f seconds\n",
ass_testAcc(trainedTree.predictFcn(testSe
t),source), toc);
tic
fprintf("Accuracy on test data for
Logistics = %f , done in %f seconds\n",
```

```
ass_testAcc(trainedLog.predictFcn(testSet
),source), toc);
tic
fprintf("Accuracy on test data for SVM =
%f , done in %f seconds\n",
ass_testAcc(trainedsvm.predictFcn(testSet
),source), toc);
tic
fprintf("Accuracy on test data for
Boosted Trees = %f , done in %f
seconds\n",
ass_testAcc(trainedBoosted.predictFcn(tes
tSet),source), toc);

fprintf("Testing Done! \n");
```

B. ADDENDUM II: DATASET IMPORT FUNCTION

```
function assprocessed =
importfile1(workbookFile, sheetName,
dataLines)
%IMPORTFILE1 Import data from a
spreadsheet
%  ASSPROCESSED = IMPORTFILE1(FILE) reads
data from the first worksheet
%  in the Microsoft Excel spreadsheet
file named FILE.  Returns the data
%  as a table.
%
%  ASSPROCESSED = IMPORTFILE1(FILE,
SHEET) reads from the specified
%  worksheet.
%
%  ASSPROCESSED = IMPORTFILE1(FILE,
SHEET, DATALINES) reads from the
%  specified worksheet for the specified
row interval(s). Specify
%  DATALINES as a positive scalar integer
or a N-by-2 array of positive
%  scalar integers for dis-contiguous row
intervals.
%
%  Example:
%  assprocessed =
importfile1("C:\Users\W20015790\OneDrive
- Northumbria University - Production
Azure AD\IS\ASSG\ass_processed.xlsx",
"TrainingSet", [2, 38798]);
%
%  See also READTABLE.
%
% Auto-generated by MATLAB on 02-May-2022
11:42:00

%% Input handling

% If no sheet is specified, read first
sheet
if nargin == 1 || isempty(sheetName)
    sheetName = 1;
end

% If row start and end points are not
specified, define defaults
if nargin <= 2
    dataLines = [2, 38798];
end

%% Set up the Import Options and import
the data
opts =
spreadsheetImportOptions("NumVariables",
14);

% Specify sheet and range
opts.Sheet = sheetName;
opts.DataRange = "A" + dataLines(1, 1) +
":N" + dataLines(1, 2);

% Specify column names and types
opts.VariableNames = ["HeartDisease",
"BMI", "Smoking", "AlcoholDrinking",
"Stroke", "PhysicalHealth",
"MentalHealth", "DiffWalking",
"AgeCategory", "Diabetic",
"PhysicalActivity", "GenHealth",
"SleepTime", "Asthma"];
opts.VariableTypes = ["categorical",
"double", "categorical", "categorical",
"categorical", "double", "double",
"categorical", "categorical",
"categorical", "categorical",
"categorical", "double", "categorical"];

% Specify variable properties
opts = setvaropts(opts, ["HeartDisease",
"Smoking", "AlcoholDrinking", "Stroke",
"DiffWalking", "AgeCategory", "Diabetic",
"PhysicalActivity", "GenHealth",
"Asthma"], "EmptyFieldRule", "auto");

% Import the data
assprocessed = readtable(workbookFile,
opts, "UseExcel", false);

for idx = 2:size(dataLines, 1)
    opts.DataRange = "A" + dataLines(idx,
1) + ":N" + dataLines(idx, 2);
    tb = readtable(workbookFile, opts,
"UseExcel", false);
    assprocessed = [assprocessed; tb];
%#ok<AGROW>
end
```

```
end
```

C. ADDENDUM III: DECISION TREE TRAINING FUNCTION

```
function [trainedTree,
validationAcc_tree] =
trainClassifier(trainingData)
% [trainedTree, validationAcc_tree] =
trainClassifier(trainingData)
% Returns a trained classifier and its
accuracy. This code recreates the
% classification model trained in
Classification Learner app. Use the
% generated code to automate training the
same model with new data, or to
% learn how to programmatically train
models.
%
%  Input:
%      trainingData: A table containing
the same predictor and response
%       columns as those imported into
the app.
%
%  Output:
%      trainedTree: A struct containing
the trained classifier. The
%       struct contains various fields
with information about the trained
%       classifier.
%
%      trainedTree.predictFcn: A function
to make predictions on new
%       data.
%
%      validationAcc_tree: A double
containing the accuracy in percent. In
%       the app, the History list
displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with
new data. To retrain your
% classifier, call the function from the
command line with your original
% data or new data as the input argument
trainingData.
%
% For example, to retrain a classifier
trained with the original data set
% T, enter:
%   [trainedTree, validationAcc_tree] =
trainClassifier(T)
%
% To make predictions with the returned
'trainedTree' on new data T2,
% use
%   yfit = trainedTree.predictFcn(T2)
%
% T2 must be a table containing at least
the same predictor columns as used
% during training. For details, enter:
%   trainedTree.HowToPredict

% Auto-generated by MATLAB on 02-May-2022
13:16:12



% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Train a classifier
% This code specifies all the classifier
options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', categorical({'No';
'Yes'}));

% Create the result struct with predict
function
predictorExtractionFcn = @(t) t(:,
predictorNames);
treePredictFcn = @(x)
predict(classificationTree, x);
trainedTree.predictFcn = @(x)
treePredictFcn(predictorExtractionFcn(x))
;

% Add additional fields to the result
struct
```

```
trainedTree.RequiredVariables =
{'AgeCategory', 'AlcoholDrinking',
'Asthma', 'BMI', 'Diabetic',
'DiffWalking', 'GenHealth',
'MentalHealth', 'PhysicalActivity',
'PhysicalHealth', 'SleepTime', 'Smoking',
'Stroke'};
trainedTree.ClassificationTree =
classificationTree;
trainedTree.About = 'This struct is a
trained model exported from
Classification Learner R2021a.';
trainedTree.HowToPredict = sprintf('To
make predictions on a new table, T, use:
\n  yfit = c.predictFcn(T) \nreplacing
''c'' with the name of the variable that
is this struct, e.g. ''trainedModel''. \n
\nThe table, T, must contain the
variables returned by: \n
c.RequiredVariables \nVariable formats
(e.g. matrix/vector, datatype) must match
the original training data. \nAdditional
variables are ignored. \n \nFor more
information, see <a
href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspac
e'')">How to predict using an exported
model</a>.');

% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Perform cross-validation
partitionedModel =
crossval(trainedTree.ClassificationTree,
'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores]
= kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAcc_tree = 1 -
kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');
```

D. ADDENDUM IV: LOGISTIC REGRESSION TRAINING FUNCTION

```
function [trainedLog, validationAcc_log]
= trainClassifier(trainingData)
% [trainedLog, validationAcc_log] =
trainClassifier(trainingData)
% Returns a trained classifier and its
accuracy. This code recreates the
% classification model trained in
Classification Learner app. Use the
% generated code to automate training the
same model with new data, or to
% learn how to programmatically train
models.
%
%  Input:
%      trainingData: A table containing
the same predictor and response
%       columns as those imported into
the app.
%
%  Output:
%      trainedLog: A struct containing
the trained classifier. The
%       struct contains various fields
with information about the trained
%       classifier.
%
%      trainedLog.predictFcn: A function
to make predictions on new
%       data.
%
%      validationAcc_log: A double
containing the accuracy in percent. In
%       the app, the History list
displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with
new data. To retrain your
% classifier, call the function from the
command line with your original
% data or new data as the input argument
trainingData.
%
% For example, to retrain a classifier
trained with the original data set
% T, enter:
%   [trainedLog, validationAcc_log] =
trainClassifier(T)
%
```

```
% To make predictions with the returned
'trainedLog' on new data T2,
% use
%   yfit = trainedLog.predictFcn(T2)
%
% T2 must be a table containing at least
the same predictor columns as used
% during training. For details, enter:
%   trainedLog.HowToPredict

% Auto-generated by MATLAB on 02-May-2022
13:18:08



% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Train a classifier
% This code specifies all the classifier
options and trains the classifier.
% For logistic regression, the response
values must be converted to zeros
% and ones because the responses are
assumed to follow a binomial
% distribution.
% 1 or true = 'successful' class
% 0 or false = 'failure' class
% NaN - missing response.
successClass = 'No';
failureClass = 'Yes';
% Compute the majority response class. If
there is a NaN-prediction from
% fitglm, convert NaN to this majority
class label.
numSuccess = sum(response ==
successClass);
numFailure = sum(response ==
failureClass);
if numSuccess > numFailure
    missingClass = successClass;
else
    missingClass = failureClass;
end
responseCategories = {successClass,
failureClass};
successFailureAndMissingClasses =
categorical({successClass; failureClass;
missingClass}, responseCategories);
isMissing = isundefined(response);
zeroOneResponse =
double(ismember(response, successClass));
zeroOneResponse(isMissing) = NaN;
% Prepare input arguments to fitglm.
concatenatedPredictorsAndResponse =
[predictors, table(zeroOneResponse)];
% Train using fitglm.
GeneralizedLinearModel = fitglm(...
    concatenatedPredictorsAndResponse,
...
    'Distribution', 'binomial', ...
    'link', 'logit');

% Convert predicted probabilities to
predicted class labels and scores.
convertSuccessProbsToPredictions = @(p)
successFailureAndMissingClasses(
~isnan(p).*( (p<0.5) + 1 ) + isnan(p)*3
);
returnMultipleValuesFcn = @(varargin)
varargin{1:max(1,nargout)};
scoresFcn = @(p) [p, 1-p];
predictionsAndScoresFcn = @(p)
returnMultipleValuesFcn(
convertSuccessProbsToPredictions(p),
scoresFcn(p) );

% Create the result struct with predict
function
predictorExtractionFcn = @(t) t(:,
predictorNames);
logisticRegressionPredictFcn = @(x)
predictionsAndScoresFcn(
predict(GeneralizedLinearModel, x) );
trainedLog.predictFcn = @(x)
logisticRegressionPredictFcn(predictorExt
ractionFcn(x));

% Add additional fields to the result
struct
trainedLog.RequiredVariables =
{'AgeCategory', 'AlcoholDrinking',
'Asthma', 'BMI', 'Diabetic',
'DiffWalking', 'GenHealth',
'MentalHealth', 'PhysicalActivity',
'PhysicalHealth', 'SleepTime', 'Smoking',
'Stroke'};
trainedLog.GeneralizedLinearModel =
GeneralizedLinearModel;
trainedLog.SuccessClass = successClass;
trainedLog.FailureClass = failureClass;
trainedLog.MissingClass = missingClass;
```

```
trainedLog.ClassNames = {successClass;
failureClass};
trainedLog.About = 'This struct is a
trained model exported from
Classification Learner R2021a.';
trainedLog.HowToPredict = sprintf('To
make predictions on a new table, T, use:
\n  yfit = c.predictFcn(T) \nreplacing
''c'' with the name of the variable that
is this struct, e.g. ''trainedModel''. \n
\nThe table, T, must contain the
variables returned by: \n
c.RequiredVariables \nVariable formats
(e.g. matrix/vector, datatype) must match
the original training data. \nAdditional
variables are ignored. \n \nFor more
information, see <a
href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspac
e'')">How to predict using an exported
model</a>.');

% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Perform cross-validation
KFolds = 5;
cvp = cvpartition(response, 'KFold',
KFolds);
% Initialize the predictions to the
proper sizes
validationPredictions = response;
numObservations = size(predictors, 1);
numClasses = 2;
validationScores = NaN(numObservations,
numClasses);
for fold = 1:KFolds
    trainingPredictors =
predictors(cvp.training(fold), :);
    trainingResponse =
response(cvp.training(fold), :);
    foldIsCategoricalPredictor =
isCategoricalPredictor;

    % Train a classifier
    % This code specifies all the
classifier options and trains the
classifier.
    % For logistic regression, the
response values must be converted to
zeros
    % and ones because the responses are
assumed to follow a binomial
    % distribution.
    % 1 or true = 'successful' class
    % 0 or false = 'failure' class
    % NaN - missing response.
    successClass = 'No';
    failureClass = 'Yes';
    % Compute the majority response
class. If there is a NaN-prediction from
    % fitglm, convert NaN to this
majority class label.
    numSuccess = sum(trainingResponse ==
successClass);
    numFailure = sum(trainingResponse ==
failureClass);
    if numSuccess > numFailure
        missingClass = successClass;
    else
        missingClass = failureClass;
    end
    responseCategories = {successClass,
failureClass};
    successFailureAndMissingClasses =
categorical({successClass; failureClass;
missingClass}, responseCategories);
    isMissing =
isundefined(trainingResponse);
    zeroOneResponse =
double(ismember(trainingResponse,
successClass));
    zeroOneResponse(isMissing) = NaN;
    % Prepare input arguments to fitglm.
    concatenatedPredictorsAndResponse =
[trainingPredictors,
table(zeroOneResponse)];
    % Train using fitglm.
    GeneralizedLinearModel = fitglm(...

concatenatedPredictorsAndResponse, ...
        'Distribution', 'binomial', ...
        'link', 'logit');

    % Convert predicted probabilities to
predicted class labels and scores.
    convertSuccessProbsToPredictions =
@(p) successFailureAndMissingClasses(
```

```
~isnan(p).*( (p<0.5) + 1 ) + isnan(p)*3
);
    returnMultipleValuesFcn = @(varargin)
varargin{1:max(1,nargout)};
    scoresFcn = @(p) [p, 1-p];
    predictionsAndScoresFcn = @(p)
returnMultipleValuesFcn(
convertSuccessProbsToPredictions(p),
scoresFcn(p) );

    % Create the result struct with
predict function
    logisticRegressionPredictFcn = @(x)
predictionsAndScoresFcn(
predict(GeneralizedLinearModel, x) );
    validationPredictFcn = @(x)
logisticRegressionPredictFcn(x);

    % Add additional fields to the result
struct

    % Compute validation predictions
    validationPredictors =
predictors(cvp.test(fold), :);
    [foldPredictions, foldScores] =
validationPredictFcn(validationPredictors
);

    % Store predictions in the original
order
    validationPredictions(cvp.test(fold),
:) = foldPredictions;
    validationScores(cvp.test(fold), :) =
foldScores;
end

% Compute validation accuracy
correctPredictions =
(validationPredictions == response);
isMissing = ismissing(response);
correctPredictions =
correctPredictions(~isMissing);
validationAcc_log =
sum(correctPredictions)/length(correctPre
dictions);
```

E. ADDENDUM V: SVM TRAINING FUNCTION

```
function [trainedsvm, validationAcc_svm]
= trainClassifier(trainingData)
% [trainedsvm, validationAcc_svm] =
trainClassifier(trainingData)
% Returns a trained classifier and its
accuracy. This code recreates the
% classification model trained in
Classification Learner app. Use the
% generated code to automate training the
same model with new data, or to
% learn how to programmatically train
models.
%
%  Input:
%      trainingData: A table containing
the same predictor and response
%       columns as those imported into
the app.
%
%  Output:
%      trainedsvm: A struct containing
the trained classifier. The
%       struct contains various fields
with information about the trained
%       classifier.
%
%      trainedsvm.predictFcn: A function
to make predictions on new
%       data.
%
%      validationAcc_svm: A double
containing the accuracy in percent. In
%       the app, the History list
displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with
new data. To retrain your
% classifier, call the function from the
command line with your original
% data or new data as the input argument
trainingData.
%
% For example, to retrain a classifier
trained with the original data set
% T, enter:
%   [trainedsvm, validationAcc_svm] =
trainClassifier(T)
%
% To make predictions with the returned
'trainedsvm' on new data T2,
% use
%   yfit = trainedsvm.predictFcn(T2)
%
% T2 must be a table containing at least
the same predictor columns as used
% during training. For details, enter:
%   trainedsvm.HowToPredict

% Auto-generated by MATLAB on 02-May-2022
13:19:54



% Extract predictors and response
% This code processes the data into the
right shape for training the
```

```
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Train a classifier
% This code specifies all the classifier
options and trains the classifier.
classificationSVM = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'linear', ...
    'PolynomialOrder', [], ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', categorical({'No';
'Yes'}));

% Create the result struct with predict
function
predictorExtractionFcn = @(t) t(:,
predictorNames);
svmPredictFcn = @(x)
predict(classificationSVM, x);
trainedsvm.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result
struct
trainedsvm.RequiredVariables =
{'AgeCategory', 'AlcoholDrinking',
'Asthma', 'BMI', 'Diabetic',
'DiffWalking', 'GenHealth',
'MentalHealth', 'PhysicalActivity',
'PhysicalHealth', 'SleepTime', 'Smoking',
'Stroke'};
trainedsvm.ClassificationSVM =
classificationSVM;
trainedsvm.About = 'This struct is a
trained model exported from
Classification Learner R2021a.';
trainedsvm.HowToPredict = sprintf('To
make predictions on a new table, T, use:
\n  yfit = c.predictFcn(T) \nreplacing
''c'' with the name of the variable that
is this struct, e.g. ''trainedModel''. \n
\nThe table, T, must contain the
variables returned by: \n
c.RequiredVariables \nVariable formats
(e.g. matrix/vector, datatype) must match
the original training data. \nAdditional
variables are ignored. \n \nFor more
information, see <a
href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspac
e'')">How to predict using an exported
model</a>.');

% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Perform cross-validation
partitionedModel =
crossval(trainedsvm.ClassificationSVM,
'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores]
= kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAcc_svm = 1 -
kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');
```

F. ADDENDUM VI: BOOSTED TREE TRAINING FUNCTION

```
function [trainedBoosted,
validationAcc_boosted] =
trainClassifier(trainingData)
% [trainedBoosted, validationAcc_boosted]
= trainClassifier(trainingData)
% Returns a trained classifier and its
accuracy. This code recreates the
% classification model trained in
Classification Learner app. Use the
```

```
% generated code to automate training the
same model with new data, or to
% learn how to programmatically train
models.
%
%  Input:
%      trainingData: A table containing
the same predictor and response
%       columns as those imported into
the app.
%
%  Output:
%      trainedBoosted: A struct
containing the trained classifier. The
%       struct contains various fields
with information about the trained
%       classifier.
%
%      trainedBoosted.predictFcn: A
function to make predictions on new
%       data.
%
%      validationAcc_boosted: A double
containing the accuracy in percent. In
%       the app, the History list
displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with
new data. To retrain your
% classifier, call the function from the
command line with your original
% data or new data as the input argument
trainingData.
%
% For example, to retrain a classifier
trained with the original data set
% T, enter:
%   [trainedBoosted,
validationAcc_boosted] =
trainClassifier(T)
%
% To make predictions with the returned
'trainedBoosted' on new data T2,
% use
%   yfit = trainedBoosted.predictFcn(T2)
%
% T2 must be a table containing at least
the same predictor columns as used
% during training. For details, enter:
%   trainedBoosted.HowToPredict

% Auto-generated by MATLAB on 02-May-2022
13:20:07


% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Train a classifier
% This code specifies all the classifier
options and trains the classifier.
template = templateTree(...
    'MaxNumSplits', 20);
classificationEnsemble = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'AdaBoostM1', ...
    'NumLearningCycles', 30, ...
    'Learners', template, ...
    'LearnRate', 0.1, ...
    'ClassNames', categorical({'No';
'Yes'}));

% Create the result struct with predict
function
predictorExtractionFcn = @(t) t(:,
predictorNames);
ensemblePredictFcn = @(x)
predict(classificationEnsemble, x);
trainedBoosted.predictFcn = @(x)
ensemblePredictFcn(predictorExtractionFcn
(x));

% Add additional fields to the result
struct
trainedBoosted.RequiredVariables =
{'AgeCategory', 'AlcoholDrinking',
'Asthma', 'BMI', 'Diabetic',
'DiffWalking', 'GenHealth',
'MentalHealth', 'PhysicalActivity',
'PhysicalHealth', 'SleepTime', 'Smoking',
'Stroke'};
trainedBoosted.ClassificationEnsemble =
classificationEnsemble;
trainedBoosted.About = 'This struct is a
trained model exported from
Classification Learner R2021a.';
trainedBoosted.HowToPredict = sprintf('To
make predictions on a new table, T, use:
```

```
\n  yfit = c.predictFcn(T) \nreplacing
''c'' with the name of the variable that
is this struct, e.g. ''trainedModel''. \n
\nThe table, T, must contain the
variables returned by: \n
c.RequiredVariables \nVariable formats
(e.g. matrix/vector, datatype) must match
the original training data. \nAdditional
variables are ignored. \n \nFor more
information, see <a
href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspac
e'')">How to predict using an exported
model</a>.');

% Extract predictors and response
% This code processes the data into the
right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'BMI', 'Smoking',
'AlcoholDrinking', 'Stroke',
'PhysicalHealth', 'MentalHealth',
'DiffWalking', 'AgeCategory', 'Diabetic',
'PhysicalActivity', 'GenHealth',
'SleepTime', 'Asthma'};
predictors = inputTable(:,
predictorNames);
response = inputTable.HeartDisease;
isCategoricalPredictor = [false, true,
true, true, false, false, true, true,
true, true, true, false, true];

% Perform cross-validation
partitionedModel =
crossval(trainedBoosted.ClassificationEns
emble, 'KFold', 5);

% Compute validation predictions
[validationPredictions, validationScores]
= kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAcc_boosted = 1 -
kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');
```

G. ADDENDUM VII: TESTING PROTOCOL FUNCTION

```
function [testAcc] =
ass_testAcc(result,source)
%Compares the test dataset with result
from the model
%   Detailed explanation goes here
    correct = 0;
    input =
table2array(source(:,"HeartDisease"));
    for N = 1 : length(result)

        a = result(N);
        b = input(N);
        if (isequal(a,b))
            correct = correct + 1;
        end
    end
testAcc = correct/length(result);
end
```