Project Report for “Ai in Health”

Biomedical Engineering MSc

By:

Agapi Konstantina Liontou 1057757

Nikolaos Chrysovitsanos 1020759

Part A

1. Preprocessing of the clinical dataset

The code for the project was created and executed in Matlab version 2022b.

For part A, we were assigned to use the clinical dataset. The first task was to perform a number of preprocessing steps: convert nominal features to numerical, remove erroneous values and handle missing values. To convert nominal features, we took the following decisions:

- Frail / Pre-frail / Non-frail → 2 / 1 / 0
- Gender, F / M → 1 / 2
- Yes / No → 1 / 0
- Sees poorly/ moderate / well → 2 / 1 / 0
- Hear poorly/ moderate / well → 2 / 1 / 0
- < 5sec / > 5sec → 1 / 2
- FALSE / TRUE → 0 / 1
- Permanent/ Occasional / No sleep problem → 2 / 1 / 0
- 5 – Excellent / 4 – Good / 3 – Medium / 2 – Bad / 1 – Very bad → 5 / 4 / 3 / 2 / 1
- 5 – A lot better / 4 – A little better / 3 – About the same / 2 – A little worse / 1 – A lot worse → 5 / 4 / 3 / 2 / 1
- 5h per week / > 2h and < 5h per week / < 2h per week / No → 3 / 2 / 1 / 0
- Current smoker / Past smoker / Never Smoked → 2 / 1 / 0

To remove erroneous values, firstly we changed all nominal values that are not used in the conversion to NaN. For numerical features, we located all values equal to 999, which we consider erroneous, along with all the outlier values that indicate a wrong measurement taken using z-score of 10, meaning that we consider values that have an abnormally large difference with the average values as wrong.

To handle missing values, first we decided to remove features and subjects that have many missing values. We used a threshold of 1/5 of entries or features to be missing to completely remove the feature or patient. This way two features and 2 patients were removed. Showing the result from the code execution:

```
variable deleted: bmi_body_fat
variable deleted: lean_body_mass
patient deleted: 3024
patient deleted: 3078
```

For the rest of the missing values we opted for the following way. For each feature we calculate three different average values, each corresponding to the subjects that are classified as frail, pre-frail and non-frail. Every missing value is replaced with the corresponding average value. For nominal features, the average is rounded to the nearest integer, thus representing one of the options available for each feature.

Finally, the preprocessing is finished and the results are saved to be used in the next task. The code for the execution is presented below and is found in part_A_1_preprocessing.m:

```
%% Import the clinical dataset to a table
clinical_dataset = readtable("clinical_dataset.csv");

%% Use an array to store the numerical version of the dataset
numeric=readmatrix("clinical_dataset.csv");

%% Convertion from nominal/categorical to numerical

% columns that contain numerical data to be used later
Numerical=[4 5 6 11 12 14 15 20 21 22 23 24 25 26 27 30 31 32 34 36 37 38 39 40 43 44
47 50 51 52 53 54 55];

for i=1:height(clinical_dataset)
% Fried
    if clinical_dataset.fried(i)== "Non frail"
        numeric(i,2)=0;
    elseif clinical_dataset.fried(i)== "Pre-frail"
        numeric(i,2)=1;
    elseif clinical_dataset.fried(i)=="Frail"
        numeric(i,2)=2;
    else
        numeric(i,2)=NaN;
    end

% Gender
    if clinical_dataset.gender(i)== "F"
        numeric(i,3)=1;
    elseif clinical_dataset.gender(i)== "M"
        numeric(i,3)=2;
    else
        numeric(i,3)=NaN;
    end

% Ortho hypotension
    if clinical_dataset.ortho_hypotension(i)== "No"
        numeric(i,7)=0;
    elseif clinical_dataset.ortho_hypotension(i)== "Yes"
        numeric(i,7)=1;
    else
        numeric(i,7)=NaN;
    end

% Vision
    if clinical_dataset.vision(i)== "Sees poorly"
```

```
            numeric(i,8)=0;
        elseif clinical_dataset.vision(i)== "Sees moderately"
            numeric(i,8)=1;
        elseif clinical_dataset.vision(i)=="Sees well"
            numeric(i,8)=2;
        else
            numeric(i,8)=NaN;
        end
    % Audition
        if clinical_dataset.audition(i)== "Hears poorly"
            numeric(i,9)=0;
        elseif clinical_dataset.audition(i)== "Hears moderately"
            numeric(i,9)=1;
        elseif clinical_dataset.audition(i)=="Hears well"
            numeric(i,9)=2;
        else
            numeric(i,9)=NaN;
        end
    % weight loss
        if clinical_dataset.weight_loss(i)== "No"
            numeric(i,10)=0;
        elseif clinical_dataset.weight_loss(i)== "Yes"
            numeric(i,10)=1;
        else
            numeric(i,10)=NaN;
        end
    % Balance_single
        if clinical_dataset.balance_single(i)== "<5 sec"
            numeric(i,13)=1;
        elseif clinical_dataset.balance_single(i)== ">5 sec"
            numeric(i,13)=2;
        else
            numeric(i,13)=NaN;
        end
    % Gait_optional_binary
        if clinical_dataset.gait_optional_binary(i)== "FALSE"
            numeric(i,16)=0;
        elseif clinical_dataset.gait_optional_binary(i)== "TRUE"
            numeric(i,16)=1;
        else
            numeric(i,16)=NaN;
        end
    % Gait_speed_slower
        if clinical_dataset.gait_speed_slower(i)== "No"
            numeric(i,17)=0;
        elseif clinical_dataset.gait_speed_slower(i)== "Yes"
            numeric(i,17)=1;
        else
            numeric(i,17)=NaN;
        end

    % Grip_strength_abnormal
        if clinical_dataset.grip_strength_abnormal(i)== "No"
            numeric(i,18)=0;
        elseif clinical_dataset.grip_strength_abnormal(i)== "Yes"
```

```
            numeric(i,18)=1;
        else
            numeric(i,18)=NaN;
        end
    % Low_physical_activity
        if clinical_dataset.low_physical_activity(i)== "No"
            numeric(i,19)=0;
        elseif clinical_dataset.low_physical_activity(i)== "Yes"
            numeric(i,19)=1;
        else
            numeric(i,19)=NaN;
        end
    % Memory_complain
        if clinical_dataset.memory_complain(i)== "No"
            numeric(i,28)=0;
        elseif clinical_dataset.memory_complain(i)== "Yes"
            numeric(i,28)=1;
        else
            numeric(i,28)=NaN;
        end
    % Sleep
        if clinical_dataset.sleep(i)== "No sleep problem"
            numeric(i,29)=0;
        elseif clinical_dataset.sleep(i)== "Occasional sleep problem"
            numeric(i,29)=1;
        elseif clinical_dataset.sleep(i)=="Permanent sleep problem"
            numeric(i,29)=2;
        else
            numeric(i,29)=NaN;
        end
    % Living_alone
        if clinical_dataset.living_alone(i)== "No"
            numeric(i,33)=0;
        elseif clinical_dataset.living_alone(i)== "Yes"
            numeric(i,33)=1;
        else
            numeric(i,33)=NaN;
        end
    % Leisure_club
        if clinical_dataset.leisure_club(i)== "No"
            numeric(i,35)=0;
        elseif clinical_dataset.leisure_club(i)== "Yes"
            numeric(i,35)=1;
        else
            numeric(i,35)=NaN;
        end
    % House_suitable_participant
        if clinical_dataset.house_suitable_participant(i)== "No"
            numeric(i,41)=0;
        elseif clinical_dataset.house_suitable_participant(i)== "Yes"
            numeric(i,41)=1;
        else
            numeric(i,41)=NaN;
        end
```

```
% House_suitable_professional
    if clinical_dataset.house_suitable_professional(i)== "No"
        numeric(i,42)=0;
    elseif clinical_dataset.house_suitable_professional(i)== "Yes"
        numeric(i,42)=1;
    else
        numeric(i,42)=NaN;
    end
% Health_rate
    if clinical_dataset.health_rate(i)== "5 - Excellent"
        numeric(i,45)=5;
    elseif clinical_dataset.health_rate(i)== "4 - Good"
        numeric(i,45)=4;
    elseif clinical_dataset.health_rate(i)=="3 - Medium"
        numeric(i,45)=3;
    elseif clinical_dataset.health_rate(i)== "2 - Bad"
        numeric(i,45)=2;
    elseif clinical_dataset.health_rate(i)=="1 - Very bad"
        numeric(i,45)=1;
    else
        numeric(i,45)=NaN;
    end
% Health_rate_comparison
    if clinical_dataset.health_rate_comparison(i)== "5 - A lot better"
        numeric(i,46)=5;
    elseif clinical_dataset.health_rate_comparison(i)== "4 - A little better"
        numeric(i,46)=4;
    elseif clinical_dataset.health_rate_comparison(i)=="3 - About the same"
        numeric(i,46)=3;
    elseif clinical_dataset.health_rate_comparison(i)== "2 - A little worse"
        numeric(i,46)=2;
    elseif clinical_dataset.health_rate_comparison(i)=="1 - A lot worse"
        numeric(i,46)=1;
    else
        numeric(i,46)=NaN;
    end
% Activity_regular
    if clinical_dataset.activity_regular(i)== "No"
        numeric(i,48)=0;
    elseif clinical_dataset.activity_regular(i)== "< 2 h per week"
        numeric(i,48)=1;
    elseif clinical_dataset.activity_regular(i)=="> 2 h and < 5 h per week"
        numeric(i,48)=2;
    elseif clinical_dataset.activity_regular(i)== "> 5 h per week"
        numeric(i,48)=3;
    else
        numeric(i,48)=NaN;
    end
% Smoking
    if clinical_dataset.smoking(i)== "Never smoked"
        numeric(i,49)=0;
    elseif clinical_dataset.smoking(i)== "Past smoker (stopped at least 6 months)"
        numeric(i,49)=1;
    elseif clinical_dataset.smoking(i)=="Current smoker"
        numeric(i,49)=2;
    else
        numeric(i,49)=NaN;
    end
end

%% Removing erroneous values

% here we delete all the values equal to 999 and we implement zscore which
% measures the distance between a data point and the mean using standard
% deviations in order to erase all other wrong values

for i=Numerical
    [S,M]=std(numeric(:,i),"omitnan");
    for j=1:height(numeric)
           z_score=(numeric(j,i)-M)/S;
           if z_score>10 || z_score<-10 || numeric(j,i)==999
            numeric(j,i)=NaN; % each value that does not fit with our
           end                % ristrictions is set to be NaN
    end
end

% array and originally numerical variables that will be created after
% deleting variables and patients
new=numeric;
new_num=Numerical;
%number of rows and cols, that will be decreased when one is erased, used to
%adjust the index for the following code that deletes wrong values
rows=540;
cols=55;
cols_del=[];
% we delete the features that have more than 1/5 of the values empty and
% do not take them under consideration for the classification
for i=1:55
    if sum(isnan(numeric(:,i)))>=540/5
        del_name=convertCharsToStrings(clinical_dataset.Properties.VariableNames{i});
        cols_del = [cols_del; del_name];
        output=sprintf('variable deleted: %s', cols_del(end));
        disp(output)
        new(:,i-55+cols)=[];
        new_num(new_num==i-55+cols)=[];
        cols=cols-1;
    end
end

% we delete the patients that have more than 1/5 of the values empty and
% do not take them under consideration for the classification
for i=1:540
    if sum(isnan(numeric(i,:)))>=10
        row_id = clinical_dataset.part_id(i);
        output=sprintf('patient deleted: %d', row_id);
        disp(output)
        new(i-540+rows,:)=[];
        rows=rows-1;
    end
end
```

```
%% Handling missing values
% calculate 3 average values for each variable(column) accordind to fried
% if the variable is categorical round them to nearest integer(category)
% depending on fried value of each patiend
% use the corresponding average to fill the array
for j=1:cols
    sum0=0;
    sum1=0;
    sum2=0;
    for i=1:rows
        if ~isnan(new(i,j))
            if new(i,2)==0
                sum0=new(i,j)+sum0;
            elseif new(i,2)==1
                sum1=new(i,j)+sum1;
            else
                sum2=new(i,j)+sum2;

            end
        end
    end
    m0=sum0/sum(new(:,2)==0,"omitnan");
    m1=sum1/sum(new(:,2)==1,"omitnan");
    m2=sum2/sum(new(:,2)==2,"omitnan");
    if ~ismember(j,new_num)
          m0=round(m0);
          m1=round(m1);
          m2=round(m2);
    end
    for i=1:rows
        if isnan(new(i,j))
            if new(i,2)==0
                new(i,j)=m0;
            elseif new(i,2)==1
                new(i,j)=m1;
            else
                new(i,j)=m2;
            end
        end
    end
end

% Create final table after changes and set variable names excluding deleted
final_data=array2table(new);
names = clinical_dataset.Properties.VariableNames;
%remove names that are deleted from the table
for i = 55:-1:1
    if ismember(names(i),cols_del)
        names(i)=[];
    end
end
final_data.Properties.VariableNames=names;

save('final_data.mat','final_data');
```

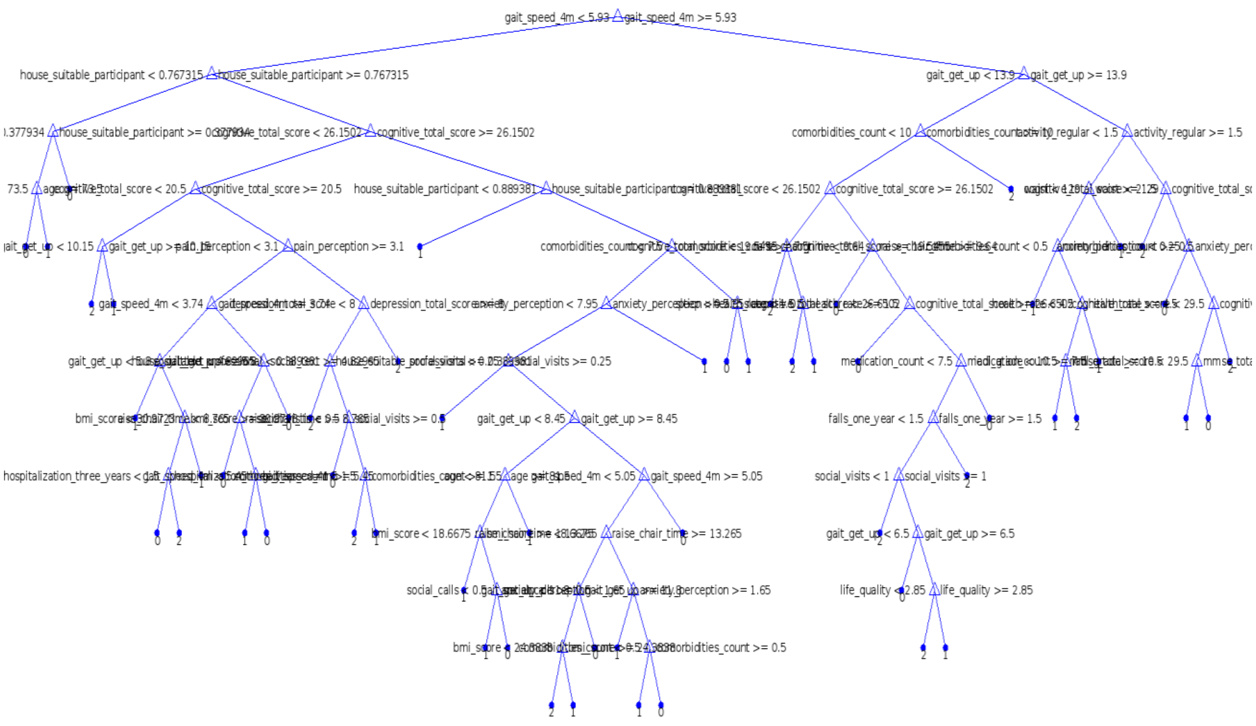
2. Classification

Using the resulting dataset, we need to perform classification analysis in order to predict the "fried" parameter. We are asked to not include in the analysis the 5 features that were used to generate the fried parameter originally. So firstly, after we load the dataset, we remove the following 5 features: weight_loss, exhaustion_score, gait_speed_slower, grip_strength_abnormal, and low_physical_activity. We also removed the part_id.

The first algorithm we used is binary Classification Tree with k-fold cross validation (k-=5). This algorithm achieved the following results, shown from matlab:

```
Accuracy of Classification Tree: 0.63 with standard deviation +/- 0.08
```

We include in our report an example graph of a created tree, although many nodes cannot be displayed clearly in the matlab environment.



The second algorithm used is Support Vector Machine (SVM), this time with holdout validation. The achieved results are the following:

```
Accuracy of SVM: 0.60
```

We want to note that during several test runs of the code SVM showed significantly bigger range of results, with accuracy in the approximate range of 0.5 to 0.75, while the tree showed much more similar results of around 0.6 to 0.65.

The code for classification is the following as found in part_A_2_classification.m:

```
load("final_data.mat");

% remove variables that are not wanted in the classification
classification_data = removevars(final_data,{'part_id','weight_loss',
'exhaustion_score', 'gait_speed_slower', 'grip_strength_abnormal',
'low_physical_activity'});

% split dataset into x(variables) and y(labels, fried parameter)
x = classification_data(:, 2:end);
y = classification_data(:, 1);


%% 1st method: Classification tree and KFold for the partition

k=5;
cv = cvpartition(538, 'KFold', k); % split the classification data
                                   % into training and test sets

accuracies = zeros(k, 1);

for i = 1:k                                  %loop through the folds and
    x_train_1 = x(cv.training(i), :);
    y_train_1 = y(cv.training(i),1);
    x_test_1 = x(cv.test(i), :);
    y_test_1 = y(cv.test(i),1);

    tree = ClassificationTree.fit(x_train_1, y_train_1); % here we train
                                                         % the model

    y_pred_1 = tree.predict(x_test_1); % prediction of the class labels for
                                       % the test set

    accuracy_1 = mean(y_pred_1 == table2array(y_test_1));
    accuracies(i) = accuracy_1;
end

mean_accuracy_1 = mean(accuracies);
std_accuracy_1 = std(accuracies);
fprintf(['Accuracy of Classification Tree: %.2f with standard deviation ' ...
         '+/- %.2f\n'], mean_accuracy_1, std_accuracy_1);

%% 2nd method: Support Vector Machine (SVM) and Holdout for partition
cv = cvpartition(538, 'Holdout', 0.2);
x_train_2 = x(cv.training, :);
y_train_2 = y(cv.training,1);
x_test_2 = x(cv.test, :);
y_test_2 = y(cv.test,1);

svm = fitcecoc(x_train_2, y_train_2);
y_pred_2 = predict(svm, x_test_2);

accuracy_2 = mean(y_pred_2 == table2array(y_test_2));
fprintf('Accuracy of SVM: %.2f\n', accuracy_2);
```

Part B

1. Preprocessing of the beacons dataset

For this part, we started with the preprocessing of the beacons dataset, containing entries with date, time and room location of every subject. We had to correct the room labels to have a dataset as homogenous as possible. We also had to remove erroneous entries from the dataset and lastly generate some new features.

We decided to perform the removal of wrong users first, as it would help by decreasing the size of the dataset. To do so, we used a simple code to locate and erase ids that were not 4-digit. This resulted in entries being removed.

Moving on to label correction, we started with isolating all the different room labels simplified to have a better outlook to the task. This code is commented out in the final script. We decided to create a dictionary of the most common rooms, and some common shortened variations of them. Thus we could find labels that are similar and change them to the desired ones, using editDistance that uses the Levenshtein distance by default. Some more scarce rooms were not considered as they do not affect the dataset much overall. The labels that will appear after the processing are Bedroom, Bathroom, Kitchen, Livingroom, Diningroom and Entrance. The dictionary also contained "Sitingroom", "Bed", "Bath", "Dinner" “Living” and "Entry", with their matches also being changed to the needed ones.

The features that we wanted to generate are the percentage of time each user spent in the 4 following rooms: Bedroom, Bathroom, Kitchen and Livingroom. To do this we measure the time spent in these 4 rooms for every user and then we calculate the percentage. We assume that we do not take into account the last entry of a given date/user because we cannot calculate the elapsed time. We create a new table containing the IDs and the 4 percentages as features. Moreover, in order to do that, we sorted our data according to part_id, date, time and we managed to have all the entries of each user grouped together.

2. Merging the two preprocessed datasets

For this task we had to merge the preprocessed clinical and beacons datasets into one. The resulting dataset must contain one entry for each person for which there are both clinical and beacons data. We created the combined dataset that contained only users that appear in both datasets. As this step can be conducted easily, it is included at the end of the preprocessing code. The code is presented below and can be found in part_B_1_2_preprocessing_merging.m:

```
%% Import the beacons dataset
beacons_dataset = readtable("beacons_dataset.csv");
beacons = beacons_dataset;
load("final_data.mat");

%% Delete all the wrong ids
for i=height(beacons):-1:1

    id=str2double(beacons.part_id(i));

    if isnan(id) || id<1000 || id>9999 % if part_id is not a 4digit number
                                       % erase it from the dataset
        beacons(i,:)=[];
    end
end

% %% Homogenous
% % do some preprocessing of the data in order to find the different types of
% % rooms that exist in the dataset and use that knowledge for the dictionary
% % that we make in the next section
%
% rooms=string.empty; % initialize a string named rooms
% counter=0;          % initialize a counter
%
% for i=1:height(beacons_dataset)
%
%    check=beacons_dataset.room(i);
%    check=lower(check);  % make uppercase to lowercase
%    check=regexprep(check,'[^a-z]',''); % delete any other character
%     if ~ismember(check,rooms) % if this room is not alreardy read
%        rooms=[rooms,check];   % place it to the rooms
%        counter=counter+1;
%     end
% end

%% Make the room labels as homegenous as possible
% make a dictionary with the basic room labels that we consider as correct
% and as they are found by the code in comments above
% Bed, Bath etc. are used to detect shortened variations
% and they will also be changed

dictionary = ["Bedroom" "Bathroom" "Kitchen" "Livingroom" "Sitingroom" "Bed" "Bath"
"Diningroom" "Dinner" "Entry" "Entrance" "Living"];

% The final table should have only one name for Bedroom, Bathroom, Kitchen,
Livingroom, Diningroom, Entrance
% less common rooms are not changed as they do not present many variations
for i=1:height(beacons)

    check=beacons.room(i);
    check=lower(check);                  % make all uppercase to lowercase
    check=regexprep(check,'[^a-z]',''); % delete any character not contained
                                         % within the range of a-z including
                                         % characters like -
    min_distance = Inf;
```

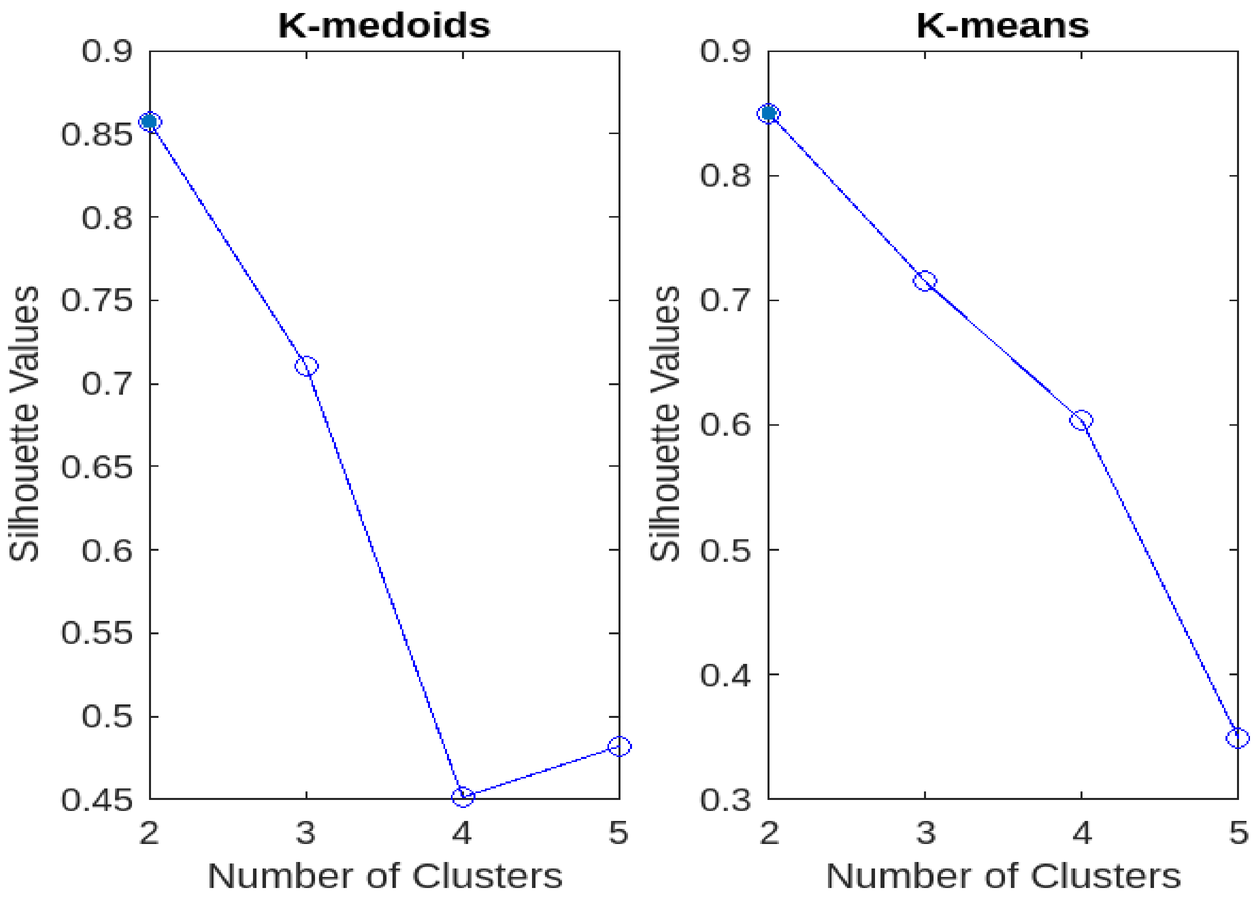
```
        correct_spelling = '';

        for k = 1:length(dictionary)

            dict=lower(dictionary(k)); % make all words in the dictionary with lowercase

            distance = editDistance(check, dict); % find edit distance between 2 strings

            if distance < min_distance
                min_distance = distance;
                correct_spelling = dictionary(k);
            end
        end

        if min_distance <= 2

        % Change shortened variation
            if correct_spelling=="Bed"
                correct_spelling=dictionary(1);
            elseif correct_spelling=="Bath"
                correct_spelling=dictionary(2);
            elseif correct_spelling=="Sitingroom"
                correct_spelling=dictionary(4);
            elseif correct_spelling=="Dinner"
                correct_spelling=dictionary(8);
            elseif correct_spelling=="Entry"
                correct_spelling=dictionary(11);
            elseif correct_spelling=="Living"
                correct_spelling=dictionary(4);
            end
            beacons.room(i)=cellstr(correct_spelling);
        end
    end

    %% Generate features

    total_time = duration(0,0,0);       % initialize the total time the patient
                                        % spent in all the rooms

    kitchen_time = duration(0,0,0);     % initialize time spent in kitchen
    bathroom_time = duration(0,0,0);    % initialize time spent in bathroom
    bedroom_time = duration(0,0,0);     % initialize time spent in bedroom
    livingroom_time = duration(0,0,0);  % initialize time spent in livingroom

    new_beacons=sortrows(beacons);          % sort our dataset first according
                                            % to the part_id, then ts_date,
                                            % then ts_time

    unique_ids=unique(new_beacons.part_id); % returns the same values as in beacons
                                            % dataset but with no repititions

    new_features=zeros(length(unique_ids),5); %array for generated features
    ids=0;

    for i=1:height(new_beacons)-1
```

```matlab
    id=new_beacons.part_id(i);

    % Detect change in patient
    if ~isequal(id,new_beacons.part_id(i+1))
        % Add generated features to new array, if no time can be calculated
        % we just put zeroes for the patient, this happens for only one
        % entry per patient per day
        ids=ids+1;
        new_features(ids,1) = str2double(cell2mat(id));

        if total_time==0
            new_features(ids,5) = 0;
            new_features(ids,3) = 0;
            new_features(ids,2) = 0;
            new_features(ids,4) = 0;
        else
            new_features(ids,5) = kitchen_time/total_time * 100;
            new_features(ids,3) = bathroom_time/total_time * 100;
            new_features(ids,2) = bedroom_time/total_time * 100;
            new_features(ids,4) = livingroom_time/total_time * 100;
        end
        % Reset counters for the next patient and move on
        total_time = duration(0,0,0);
        kitchen_time = duration(0,0,0);
        bathroom_time = duration(0,0,0);
        bedroom_time = duration(0,0,0);
        livingroom_time = duration(0,0,0);
        continue;
    end

   % for the last entry of a day, we cant be sure of the time spent
   % in the room so we do not consider it
    if ~isequal(new_beacons.ts_date(i),new_beacons.ts_date(i+1))
    continue;

% This commented part would add the total days passed between entries
% if we wanted to consider it in the calculations
%     date = num2str(new_beacons.ts_date(i));
%     day = str2double(date(end-1:end));
%     month = str2double(date(end-3:end-2));
%     year = str2double(date(1:end-4));
%     nextdate = num2str(new_beacons.ts_date(i+1));
%     nextday = str2double(nextdate(end-1:end));
%     nextmonth = str2double(nextdate(end-3:end-2));
%     nextyear = str2double(nextdate(1:end-4));
%     days = nextday-day + (nextmonth-month)*30 + (nextyear-year)*365;

    end

 % add the calculated time spent in the room to the total time and to the
 % coresponding counter for the room
     added_time = new_beacons.ts_time(i+1) - new_beacons.ts_time(i);

     total_time = total_time + added_time ;
     if new_beacons.room(i)=="Kitchen"
```

```
            kitchen_time = kitchen_time + added_time;
         elseif new_beacons.room(i)=="Bathroom"
            bathroom_time = bathroom_time + added_time;
         elseif new_beacons.room(i)=="Bedroom"
            bedroom_time = bedroom_time + added_time;
         elseif new_beacons.room(i)=="Livingroom"
            livingroom_time = livingroom_time + added_time;
         end
end

% Final calculation for the last patient that could not be done in the loop
% for the last entry we cant measure the time so we dont add it
ids=ids+1;
if total_time==0
    new_features(ids,5) = 0;
    new_features(ids,3) = 0;
    new_features(ids,2) = 0;
    new_features(ids,4) = 0;
else
    new_features(ids,1) = str2double(cell2mat(id));
    new_features(ids,5) = kitchen_time/total_time * 100;
    new_features(ids,3) = bathroom_time/total_time * 100;
    new_features(ids,2) = bedroom_time/total_time * 100;
    new_features(ids,4) = livingroom_time/total_time * 100;
end

% turn final generated feature into a table
percentages=array2table(new_features,'VariableNames',{'part_id','Bedroom_time','Bathr
oom_time','Livingroom_time','Kitchen_time'});


%% Merging clinical and beacons dataset (the preprocessed ones)
% the merged dataset only contains patients that appeared in both datasets
merged_dataset=innerjoin(final_data,percentages);

save('final_beacons.mat','beacons');
save('merged_dataset.mat','merged_dataset');
```

3. Clustering

Finally, we performed clustering to the merged dataset and used internal criteria for evaluation. For start, we picked the k-medoids and k-means algorithms that have very similar execution and compared their evaluation with Silhouette index for 2 to 5 clusters. The following figures show the result of the evaluation. They achieve the best score for 2 clusters, followed by 3.

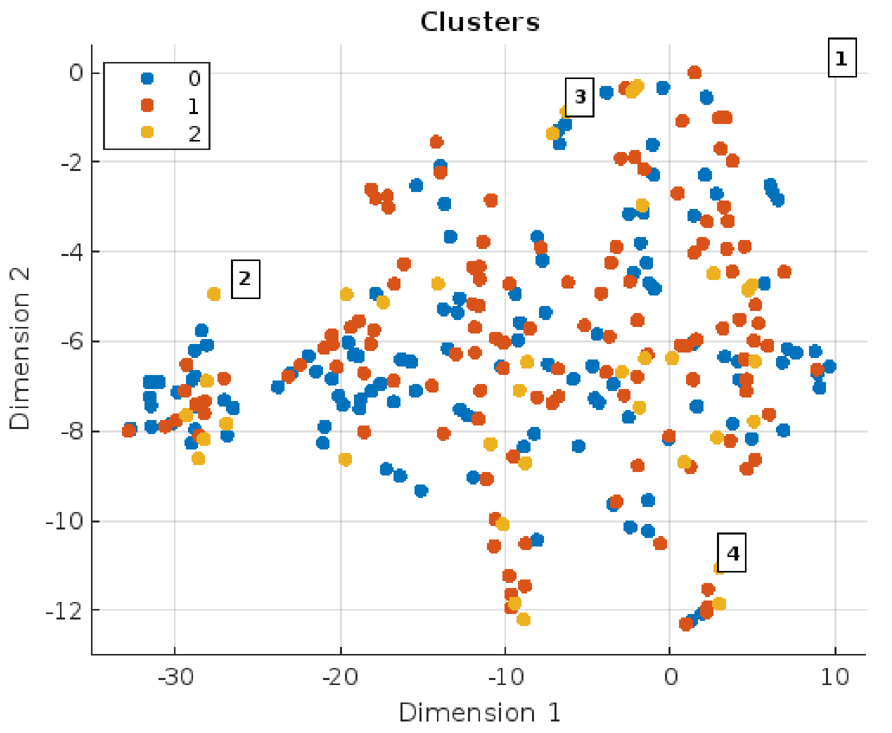


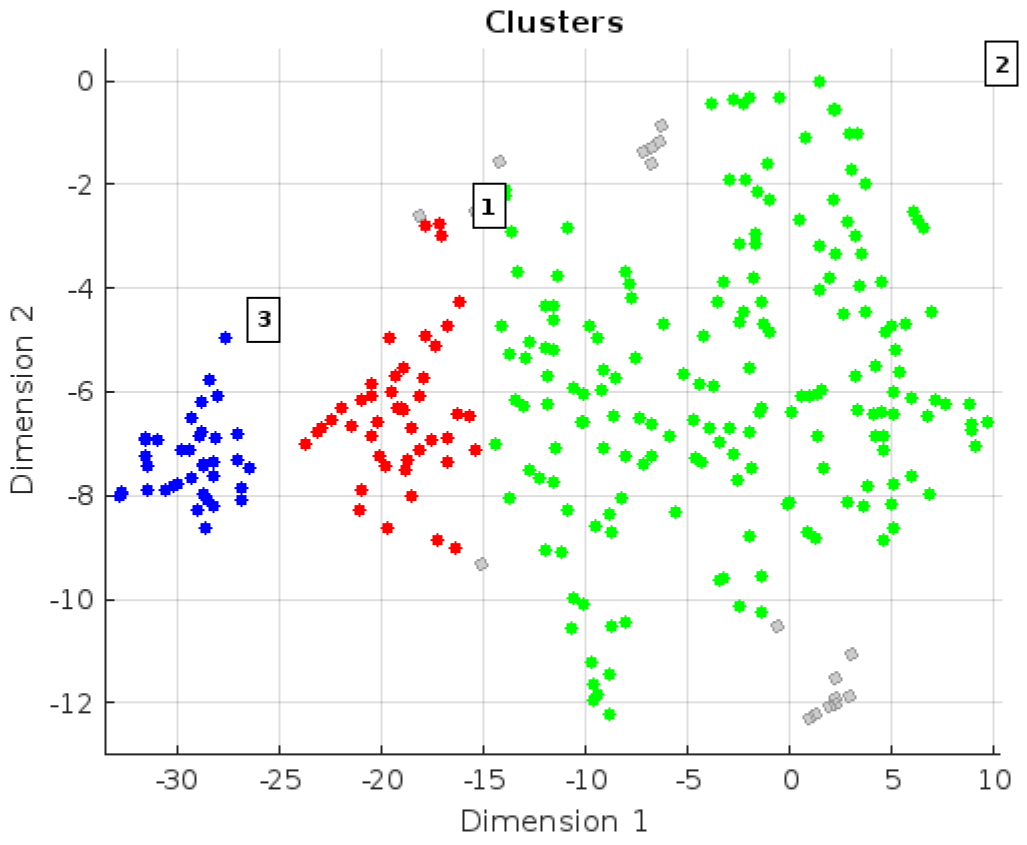
max k-medoids evaluation score: 0.857392

max k-means evaluation score: 0.850146

As an extra step, we experimented with dimensionality reduction and performed t-SNE in the dataset to perform clustering with DBscan. The 2 figures below show the results, the first one is the resulting 2-dimensional dataset with labels according to “fried” parameter and the second one is the result of dbscan. Performing evaluation with DaviesBouldin criterion we get:

```
dbscan evaluation score: 0.747353
```


Clusters
0
1
2
1
2
3
4
Dimension 1
Dimension 2


Clusters
1
2
3
Dimension 1
Dimension 2

The code for the clustering step is written below and can be found in part_B_3_clustering.m:

```
load("merged_dataset.mat");

% create array with the merged dataset from the preprocessing step
merged_array=table2array(merged_dataset);

% We do not consider the ids and the fried parameter, since it is already
% used for classification in part A
clustering_data=merged_array(:,3:end);

%% Clustering with Kmedoids

%perform kmedoids for variable number of clusters
for i=1:5
    idx_1(:,i)=kmedoids(clustering_data,i);
end

%evaluate k-medoids with silhouette internal criterium
eva_1=evalclusters(clustering_data,idx_1,'silhouette');

% plot graph for evaluation score by number of clusters
subplot(2,2,1)
plot(eva_1)
title('K-medoids')
fprintf('max k-medoids evaluation score: %f\n', max(eva_1.CriterionValues));

%% Clustering with Kmeans

% do the same as with k-medoids
for i=1:5
    idx_2(:,i)=kmeans(clustering_data,i);
end

eva_2=evalclusters(clustering_data,idx_2,'silhouette');
subplot(2,2,2)
plot(eva_2)
title('K-means')
fprintf('max k-means evaluation score: %f', max(eva_2.CriterionValues));

%% Clustering with DBscan

% use TSNE for dimensionality reduction, making all the points 2d.
rng default
Y=tsne(clustering_data);

%show results
subplot(2,2,3)
gscatter(Y(:,1),Y(:,2),merged_array(:,2))

%estimate epsilon for dbscan
min_points = 5;
max_points = 20;
```

```matlab
epsilon=clusterDBSCAN.estimateEpsilon(Y,min_points,max_points)


min_points = 10;                    % change to the desired value for minPts

% run dbscan
clusterer = clusterDBSCAN('MinNumPoints',min_points,'Epsilon',epsilon);
idx_3 = clusterer(Y);

% plot the results
subplot(2,2,4)
plot(clusterer,Y,idx_3)

% remove outliers, marked with -1 from the data and the labels for
% evaluation score
data = clustering_data(idx_3 ~= -1,:);
labels = idx_3(idx_3 ~= -1);

% evaluation with DaviesBouldin Index (DBI)
eva_3=evalclusters(data,labels,'DaviesBouldin');
fprintf('dbscan evaluation score: %f', eva_3.CriterionValues);
```

Conclusions

To sum up our work for this project, we performed some preprocessing steps in the two given datasets. We used the first one for classification and we merged it with the second one for clustering. Next on, we present our thoughts on working this project, what we concluded and how we can interpret the results.

The preprocessing of the data is a necessary procedure not only to make them suitable for use in an algorithm but also to achieve the most efficient use of the given dataset. Often it is as demanding as the task we are preparing them for if not more in some cases. We found ourselves spending time and effort in even presumably simple preprocessing steps, in order to make critical decisions and avoid logic errors in our coding. We debated a lot on what we should consider a wrong value and how we should handle missing values in the first part. While in the second part of the project, correcting the room labels in the dataset and turning our ideas into code required much more thought and planning than we expected.

On executing the classification and clustering, we had to do some research and gain input to decide which algorithms we should use. Again we found ourselves having a difficult time deciding what works best. Once decided, we noticed that actually running, for example the classification, is much more straight-forward than planning for it. All that is of course thanks to the tools and the simplicity that Matlab and other modern programming environments in general have to offer.

For the classification, we used decision trees and SVM. The accuracy scores for both algorithms were not great, although SVM did manage to score more that 70% in a few runs. What was more easily concluded than the effectiveness, is how much more SVM varied in accuracy compared to the much more stable decision tree method. The use of k-fold cross validation did help with providing more consistent results for the first algorithm. A future note would be to experiment more with validation methods and generally with parameters in both classification algorithms and maybe perform SVM multiple times to attain some average accuracy score.

For clustering, we performed k-medoids, k-means and DBScan. The first two were evaluated with silhouette indexing and had similar behavior for variable number of clusters. Both methods had a better score for 2 clusters, which was found to be the optimal number, contradicting the 3 class division of the clinical dataset. It is an interesting point that stems from the transition to unsupervised learning. For the DBScan, we opted to perform some dimensionality reduction with TSNE, as it is a method not suited for high dimensionality. It provided a very interesting visualization of our problem, and showed moderate success of the DBScan method, which was evaluated with DaviesBouldin indexing. An explanation would be that this method is evidently not the optimal for this dataset and also that we did not handle the dimensionality reduction as well as we could have.

On last thoughts, we saw for ourselves in a certain extent how working on AI can be considered an art. We have to explore our options, do a lot of trial and error and be prepared to be surprised by the results. In this project we mostly aimed to get some hands-on experience on performing classification and clustering, not to forget the important part of data preprocessing. Thus we did not strive for getting optimal results. Some things we think can add to this work is trying other interpretations of the dataset and/or perform extra preprocessing steps, for example normalization of the values.